UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

INTERNAL HACKING DETECTION USING MACHINE LEARNING

Madalena da Costa França Ribeiro Guerra

MESTRADO EM CIÊNCIA DE DADOS

Dissertação orientada por:
Professora Doutora Cátia Luísa Santana Calisto Pesquita

2020

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

INTERNAL HACKING DETECTION USING MACHINE LEARNING

DISSERTAÇÃO

Madalena da Costa França Ribeiro Guerra

MESTRADO EM CIÊNCIA DE DADOS

Dissertação orientada pela Professora Doutora Cátia Luísa Santana Calisto Pesquita e supervisionado na Altice Portugal pelo Engenheiro José António dos Santos Alegria.

2020

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor Prof. Cátia Pesquita for her guidance, mentorship, and support, not only academically but also emotionally, throughout the master's journey.
I would like to thank to Eng. José Alegria for his immense knowledge and experience, meaningful and helpful advices, and for giving me the chance to work on this project. I am also very grateful to Ricardo Ramalho for his patience, continuous guidance, constant feedback, and motivation. I also want to thank everyone in the DCY department at Altice Portugal for their endless efforts in providing a very friendly and enjoyable place to work.
Last but not least, I am eternally grateful to my family and friends for their continuing support. Most of all my parents, Amílcar and Zaida, for their unconditional love, support, and encouragement.

Dedicated to my family and friends

Resumo

Numa era onde se lida com uma quantidade massiva de dados, a compreensão do seu valor através da aplicação de técnicas analíticas avançadas ganhou grande destaque nas mais diversas organizações. Nos últimos anos, os ciberataques começaram a afetar a segurança das empresas, tornando-se numa das suas principais preocupações. Esses ataques podem ser provocados por intrusos ou por pessoas mal-intencionadas. Contudo, com os ataques internos a tornarem-se cada vez mais frequentes e demonstrando-se ser mais prejudiciais do que ataques externos em grandes empresas, este tem sido o principal foco da maioria das investigações feitas nos últimos anos neste domínio.

A deteção de comportamento anómalo de utilizadores é um tópico de pesquisa de cibersegurança que se preocupa com a deteção de anomalias no comportamento dos utilizadores correspondentes quer a ataques internos de carácter malicioso, quer a atividades anómalas sem intenção maliciosa, como erros do sistema ou erros humanos. No entanto, essa deteção deve ser feita o mais rápido possível, gerando um alerta caso se trate, por exemplo, de um acesso mal-intencionado ou não autorizado, de forma a evitar ou minimizar consequências prejudiciais na empresa. Dado que milhões de logs de utilizadores no sistema são produzidos diariamente, esta tarefa ultrapassa o poder cognitivo dos analistas humanos.

Machine Learning é uma área de *Computer Science* que se encontra em constante pesquisa e cujo objetivo é a criação de algoritmos e sua correta aplicação, permitindo que as máquinas aprendam automaticamente a partir de dados relacionados com um determinado problema e, posteriormente, façam previsões. Como tal, foram apresentadas várias técnicas de deteção de comportamento anómalo de utilizadores baseadas em *Machine Learning* que se mostraram extremamente eficientes nesse domínio, principalmente abordagens não supervisionadas, pois não requerem uma classificação manual por parte de especialistas humanos que em, problemas desta natureza, é muito desafiante de se obter.

O trabalho desenvolvido nesta dissertação insere-se no domínio de cibersegurança e faz parte de um projeto abrangente do Departamento de Cibersegurança (DCY) da Altice Portugal (MEO). O principal objetivo desta dissertação é desenvolver e avaliar um sistema de deteção de comportamento anómalo de trabalhadores da empresa, suportado por técnicas baseadas *Machine Learning*, para detetar atividades ilícitas de acesso a dados sensíveis, usando tecnologias como *Elastic Search, Kibana, Python, Jupyter Notebook* e bibliotecas

Python tais como *elasticsearch, fastparquet, dask, numpy, pandas, scikit-learn, plotly* e *pyod*.

O conjunto de dados usado nesta dissertação diz respeito a uma coleção de registos de logs de acessos realizados por utilizadores num intervalo de tempo de seis meses, enriquecidos com meta-dados dos utilizadores, da respetiva conta por onde acedeu e dos objetos acedidos, em que cada registo de log tipifica o acesso de um utilizador a um determinado objeto. A diversidade e volume massivo de dados produzidos constituíram um grande desafio para o desenvolvimento deste projeto.

A deteção de comportamento anómalo por parte dos utilizadores de uma organização com o principal objetivo de detetar ameaças internas tem sido um campo de pesquisa ativa no domínio da segurança. Apesar do grande número de pesquisas sobre este tópico e da existência de vários métodos que visam a deteção de anomalias, ainda não foi verificado um consenso quanto ao conjunto de métodos mais eficiente para este propósito. Para além disso, é possível identificar várias falhas nessas investigações, como a escassez de investigações aplicadas a dados reais de empresas, e não a simulações da realidade; e a análise de desvios no comportamento do utilizador ignorando tanto o contexto empresarial, bem como as relações entre utilizadores e entidades. A adição dessa informação é essencial para uma melhor eficiência e precisão do sistema de deteção, permitindo minimizar o número de falsos positivos.

Para atingir os objetivos desta tese, foi criado e avaliado um sistema automático robusto capaz de processar e analisar um grande volume de registos de dados correspondentes a logs de atividade dos utilizadores autorizados no sistema da empresa de forma a detetar ocorrências de comportamento anómalo, principalmente, associado a uma atividade de carácter malicioso. Como tal, é essencial que o sistema seja capaz de discriminar as anomalias detetadas para, posteriormente, classificar como sendo (passível de ser) uma ameaça interna ou não, de forma a evitar ou minimizar os danos para a empresa.

O sistema desenvolvido nesta dissertação é composto por vários módulos. O primeiro módulo consiste na Seleção dos Dados e dos Atributos. Neste módulo, através da combinação de várias técnicas de visualização, disponibilizadas pelo Kibana, com o conhecimento de especialistas no domínio, é possível selecionar um foco de dados para o qual se pretende direcionar a investigação, bem como descobrir os atributos relevantes que melhor possam contribuir para a obtenção de bons resultados. O módulo seguinte é a Integração, Pré-Processamento e Preparação dos Dados. Neste módulo, os dados armazenados no ElasticSearch são, iterativamente, extraídos para um Jupyter Notebook Python sob a estrutura de dados Dask DataFrame, submetidos a um processo de limpeza, agregados e integrados num conjunto de dados com um sumário da atividade diária de cada conta de utilizador, armazenado num ficheiro de formato *parquet*. Uma vez processados e integrados todos os registos diários, é efetuada uma Transformação dos Dados. Esta componente computa inúmeros atributos correspondentes a agregações e rácios de forma a construir

um perfil caracterizador de cada conta de utilizador, com o objetivo de identificar grupos de contas semelhantes entre si. Uma vez identificados esses grupos, é criado um conjunto de dados, escalado e centrado, com a atividade diária de cada conta tendo em consideração o histórico de atividade de todas as contas, o histórico de atividade da respetiva conta, e o histórico de atividade do grupo de contas a que pertence. No módulo de Caracterização das Contas, através da comparação dos perfis de contas anteriormente criados, é possível agrupar as contas dos utilizadores com comportamento semelhante, através da aplicação de técnicas de *Clustering*, e construir um perfil para cada um desses grupos de contas. Adicionalmente, através da identificação de desvios face à baseline ou ao ”esperado”empiricamente, é possível detectar contas anómalas, isto é, contas cujo comportamento seja persistentemente anormal. O módulo seguinte é a Deteção de Anomalias, onde são detectados eventos anómalos através da aplicação de um conjunto de algoritmos de deteção de anomalias baseados em aprendizagem não supervisionada com combinações de parâmetros e de atributos distintas e, posteriormente, comparados. O desempenho de cada um desses detetores é avaliado através da injeção de dez cenários distintos de ameaças internas, criadas com a ajuda dos especialistas no domínio. O último módulo é a Caracterização das Anomalias. Este módulo permite distinguir as anomalias, não só identificando aquelas que são passíveis de se tratar de acessos ilícitos ou não, mas também identificar o tipo de atividade que a originou, através da criação de várias métricas. Esta caracterização permite que as anomalias detetadas sejam representadas num dashboard, criado no Kibana, e facilmente visualizadas pelos analistas.
O sistema desenvolvido nesta dissertação obteve resultados promissores. Este demonstrou ser capaz de detetar comportamentos anómalos em diversos domínios por parte de uma conta, tendo em consideração não só a sua atividade passada como também a atividade de outras contas semelhantes. Para além de detetar de forma robusta essas anomalias, é capaz de identificar as ações que lhes deram origem, permitindo a exibição de um conjunto de visualizações num *dashboard* já direcionadas para o foco de investigação. Deste modo, os analistas identificam facilmente a natureza e o risco de cada anomalia e encaminham-na para as respetivas entidades competentes de forma a mitigar eventuais consequências que possam prejudicar a empresa.

Palavras-chave: detecção de comportamento anómalo de utilizadores, cibersegurança, ameaça interna, aprendizagem máquina, análise do comportamento de utilizadores e entidades

Abstract

Being able to prevent and early detect insider threats through an automated forewarning system has been a massive challenge for large companies. In recent years, to fill this gap several anomaly user behavior algorithms based on machine learning have been proposed, experimentally evaluated and analyzed in numerous surveys.
The present work was conducted in the cybersecurity department (DCY) of Altice Portugal (MEO) and aims to address this problem identifying the families of unsupervised anomaly detection techniques that are more effective for insider threats detection based on a large dataset corresponding to a collection of users' access log records.
To this end, multi-domain attributes related to possible insider threats are interactively extracted and processed, creating a summary of user account's daily activity. A clustering-based algorithm that groups and characterizes similar accounts was applied. Without any example anomalies required in the training set, anomaly detection techniques were computed over those profiles, identifying unusual changes in user account behavior on a current day. Finally, to make it easier for analysts and managers to understand the anomaly, anomaly metrics and a visualization dashboard were created.
To evaluate the efficiency of this project ten insider threat scenarios were injected and was found that the system can successfully detect anomalous behavior that may be an insider threat event.

Keywords: anomalous user behavior detection, cybersecurity, insider threat, machine learning, user and entity behavior analytics

Contents

List of Figures **xvi**

List of Tables **xx**

1 Introduction **1**
1.1 Motivation 1
1.2 Objectives and Contributions 2
1.3 Document Structure 2

2 Background **5**
2.1 Anomaly Detection 5
2.2 Unsupervised Anomaly Detection Algorithms 7
2.2.1 k-Nearest Neighbors 7
2.2.2 One-Class Support Vector Machines 7
2.2.3 Local Outlier Factor 8
2.2.4 Cluster-Based Local Outlier Factor 8
2.2.5 Isolation Forest 8
2.3 Clustering 9
2.3.1 k-Means 9
2.3.2 Spectral Clustering 9
2.4 Dimensionality Reduction 10
2.4.1 Principal Components Analysis 10
2.5 Visualization and Interaction 10
2.5.1 Parquet 10
2.5.2 Kibana 10
2.6 Fundamental Tools and Libraries 11
2.6.1 Dask 11
2.6.2 PyOD 12
2.6.3 Plotly 12
2.6.4 T-Distributed Stochastic Neighbouring Entities 12

3 Related Work **13**

4 Analysis **19**
4.1 Data Description 21

5 Architecture **25**

6 Implementation **29**
6.1 Exploratory Data Analysis and Feature Selection 29
6.2 Data Integration, Preprocessing and Preparation Pipeline 32
6.3 Data Transformation 37
6.3.1 Feature Engineering for User Accounts Characterization 37
6.3.2 Feature Engineering for Anomaly Detection 38
6.4 User Accounts Characterization and Anomalous User Accounts Detection 39
6.5 Anomaly Detection Module 40
6.6 Anomaly Characterization Module 44

7 Results and Discussion **47**
7.1 Exploratory Data Analysis and Feature Selection 47
7.2 Data Integration, Preprocessing and Preparation Pipeline 53
7.3 Data Transformation 54
7.3.1 Feature Engineering for User Accounts Characterization 54
7.3.2 Feature Engineering for Anomaly Detection 55
7.4 User Accounts Characterization and Anomalous User Accounts Detection 55
7.5 Anomaly Detection Module 60
7.6 Anomaly Characterization Module 60
7.7 Detected Anomalies Examples 61
7.8 Discussion 69

8 Conclusion and Future Work **71**

A Hyper-parameter Tuning for User Account Characterization **73**

B Hyper-parameter and Weight Assignment of Features Tuning for Anomaly Detection **75**

References **91**

List of Figures

2.1 Example of a Kibana Dashboard. 11

5.1 Architecture of the Anomaly Detection System. 27

6.1 Questions that our features must answer. 30
6.2 Representative example of the relationship between a user and a user account. 31
6.3 Ordered levels of granularity characterizing the accessed object. 32
6.4 Data Pre-processing Pipeline Diagram. 33
6.5 Two-way pivot table between *actor_executable* and *actor_range_type* values with additional metrics columns. 35
6.6 Illustrative example of the results obtained after the Data Cleaning and the Data Aggregation and Feature Extraction processes. 36
6.7 Illustrative example of the result obtained after the aggregation by user account. 38
6.8 Illustrative example of the relationship between sets of information given by the features. 42
6.9 Dashboard for comparison of user accounts behavior across multiple attributes and domains. 45
6.10 Dashboard for temporal comparison of individual user account activity on multiple attributes and domains. 46

7.1 Absolute frequency distribution in logarithmic scale of features' values. . 49
7.2 Absolute frequency distribution in logarithmic scale of features' values according to user accounts. 49
7.3 Absolute frequency distribution of *ts* feature values considering distinct time granularity. 50
7.4 Stacked column plots analyzing relationships between values of two features. 51
7.5 Visualization of the results obtained by the Characterization of User Accounts Module. 53
7.6 Daily count of user accounts' access log records with the same total number of records. 54

7.7 Performance evaluation metrics for k-Means algorithms. 56
7.8 Performance evaluation metrics for Spectral Clustering algorithms. 56
7.9 Analysis of the clustering of similar user accounts obtained. 57
7.10 Box-plots to aid detection of user accounts that have persistently abnormal behavior. 58
7.11 Examples of detected user accounts that have persistently abnormal behavior. 59
7.12 Example of detected anomalies 1 - Nominal user account with abnormal access volume. 61
7.13 Example of detected anomalies 2 - Application user account with abnormal access volume. 61
7.14 Example of detected anomalies 3 - Application user account with abnormal behavior comparing to its past activity. 62
7.15 Example of detected anomalies 6 - Nominal user account with abnormal time accesses with respect to its past activity and its account group activity. 64
7.16 Example of detected anomalies 7 - Nominal user account that performed abnormal actions with respect to its past activity and its account group activity. 65
7.17 Example of detected anomalies 8 - Nominal user account with abnormal time, location and range type accesses with respect to its past activity and its account group activity. 66
7.18 Example of detected anomalies 9 - Application user account with abnormal time, location and range type accesses with respect to its past activity. 67
7.19 Example of detected anomalies 10 - Application user account with abnormal action and range type accesses with respect to its past activity and its account group activity. 68

List of Tables

3.1 Related work summary 16
3.1 Related work summary (cont.). 17
3.1 Related work summary (cont.). 18

4.1 Data Description 21
4.1 Data Description (cont.). 22
4.1 Data Description (cont.). 23

6.1 Anomaly metrics associated with each of the injected anomalies. 44

7.1 Variable Datatype Identification 47
7.1 Variable Datatype Identification (cont.). 48
7.2 Actor_range_type frequency table 48
7.3 Db_brand frequency table 48
7.4 Missing and Noisy Data Identification 51
7.4 Missing and Noisy Data Identification (cont.). 52
7.5 Number of distinct values for each feature before and after Data Cleaning. 53
7.6 Dataset size and dimensionality after running each step involved in the Data Extraction, Preprocessing and Preparation module. 54
7.7 Dataset size and dimensionality after running each step involved in the Feature Engineering for Clustering component. 55
7.8 Dataset size and dimensionality after running each step involved in the Feature Engineering for Anomaly Detection component. 55
7.9 Anomalies detected by each Anomaly Detection algorithm. 60
7.10 Feature subset dimension of each anomaly metric. 60
7.10 Feature subset dimension of each anomaly metric (cont.). 61
7.11 Example of detected anomalies 4 - Nominal user account with abnormal executable accesses with respect to its past activity. 63
7.12 Example of detected anomalies 5 - Nominal user account with abnormal time accesses with respect to its past activity. 63

A.1 K-Means Parameter Values. 73
A.2 Spectral Clustering Parameter Values. 73

A.2 Spectral Clustering Parameter Values (Cont.). 74
A.3 Spectral Clustering Parameter Values that achieved better results. 74

B.1 k-Nearest Neighbor for Anomaly Detection Parameter Values. 75
B.2 One-Class Support Vector Machines Parameter Values. 75
B.2 One-Class Support Vector Machines Parameter Values (cont.). 76
B.3 Local Outlier Factor Parameter Values. 76
B.4 Clustering Based Local Outlier Factor Parameter Values. 76
B.5 Isolation Forest Parameter Values. 77
B.6 Range of features weights investigated for each feature group. 77
B.7 Combination of features weights that achieved the best result. 77
B.7 Combination of features weights that achieved the best result (cont.). . . . 78
B.8 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result. 78
B.9 Range of values considered in max_features of the Isolation Forest algorithm for each anomaly metric. 78
B.10 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Action_anomaly. 78
B.11 Combination of Isolation Forest algorithm parameters that achieved the best result for Location_anomaly. 78
B.12 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Range_type_anomaly. 79
B.13 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Executable_anomaly. 79
B.14 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Hourly_anomaly. 79
B.14 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Hourly_anomaly (cont.). 80
B.15 Combination of Isolation Forest algorithm parameters that achieved the best result for Number_anomaly. 80
B.16 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for User_anomaly. 80
B.17 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Group_anomaly. 81
B.18 Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for All_anomaly. 81

Chapter 1

Introduction

1.1 Motivation

Over the last years, cyberattacks, especially internal hacking, have become an increasing threat and one of the major concerns in enterprise security. In order to address this issue it is important to explore the great potential of the large amount of data collected by those enterprises.

Anomalous User Behavior Detection is the cybersecurity research topic that focuses on the detection of anomalies in user behavior corresponding to both malicious insider attacks and anomalous activities without malicious intent, such as errors or human mistakes. However, this detection must be done as quickly as possible, raising an alert in the company whether for instance it is a malicious or unauthorized access to prevent or minimize company harmful consequences. Given that millions of raw system user logs are produced daily, this task scales beyond the cognitive power of human analysts.

Machine Learning is a Computer Science area on active research whose purpose is the creation of algorithms and their correct application, allowing machines to automatically learn from data related to a given problem and thereafter make predictions. As such, several anomaly user behavior detection techniques based on machine learning that proved to be extremely efficient in this domain have been presented, mostly unsupervised approaches since labeling requires a manual human expert classification.

The work developed in this dissertation focuses on the cybersecurity domain and is part of an encompassing project from the cybersecurity department (DCY) of Altice Portugal (MEO). The main goal of this dissertation is to develop and evaluate an anomalous user behavior detection system supported by techniques based on machine learning to detect illicit sensitive data access activities, using technologies such as Elastic Search, Ruby, python scikit-learn and Jupyter Notebook.

1.2 Objectives and Contributions

The main objective of this thesis is to develop and evaluate a robust automated system that is capable of processing and analyzing a large collection of activity log data in order to detect anomalies evocative of occurrences of insider threat behavior. In this way, the system must be able to characterize its detected anomalies adding the classification of either being an insider threat or not.
The main contributions of this dissertation are towards:

1. Development of a Exploratory Data Analysis and Feature Selection Module. Combining many visualization techniques, using Kibana's *Visualize*, with the expert's domain knowledge, this module selects the data and features that most contribute to the problem that is intended to be solved.

2. Development of a Data Integration, Preprocessing and Preparation Pipeline. This pipeline receives data corresponding to user account log records in a current day, processes it and integrates it in a user accounts' daily activity dataset.

3. Development of User Account and User Account Groups Profiling Module. This module computes multiple aggregation and ratio features building both user account's and user account group's behaviour profiles with a complete description of their behavior.

4. Development of User Accounts Characterization and Anomalous User Accounts Detection Module. In this component user accounts are clustered creating groups of user accounts with similar overall behaviors. Subsequently, detects user account behaviors that are persistently abnormal by identifying a significant deviation compared against those in the same account group.

5. Development of an Anomaly Detection Component. This component implements a centered and scaled dataset based on all user accounts' behavioral history, on the user account's behavioral history and the behavioral history of all user accounts in the same group. After that, it applies and compares the performance of multiple anomaly detection algorithms over the normalized scores and ranks obtained.

6. Development of an Anomaly Characterization Module. This module is able to characterize the anomalies detected in terms of different aspects (Action, Location, Range_type, Executable, Hourly, Number, User, Group, All).

1.3 Document Structure

The remainder of this dissertation is organized as follows. Chapter 2 presents the background on anomaly detection, where the problem is described in detail, as well as pre-

senting all the basic concepts and tools that are required for the implementation of the solution developed in this work. Chapter 3 summarizes previous works in which machine learning-based approaches for insider threat detection were also employed. Chapter 4 briefly describes the context in which this work is inserted, its requirements and the dataset used. The general architecture of the developed system and a brief description of each component can be found in the Chapter 5. A more detailed description of that architecture and its implementation was made in the in Chapter 6. In Chapter 7 the results of the experiments performed were presented and discussed. Finally, Chapter 8 contains concluding remarks, as well as possible future work.

Chapter 2

Background

This chapter presents an overview of the Anomaly Detection problem, where all the basic concepts related to this topic are briefly described, as well as the techniques and tools employed in the development of the solution presented in this dissertation.

2.1 Anomaly Detection

Detecting anomalies or outliers in large amounts of data has been studied in several research communities since the 19th century [24]. Although diverse anomaly detection methods have been developed, outlier detection continues to be an active research problem due its fundamental role in a wide variety of applications across many different domains such as intrusion detection [30, 7, 68], fraud detection [65, 69, 5], medical and life science anomaly detection [33, 39, 36], image processing [11, 84, 71], sensor networks [22, 1, 94], and industrial damage detection [8, 91, 93].

A substantial change of perspective on this issue has been raised from the following definition given by Grubbs in 1969: "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs" [37]. However, due their high significance, researchers have been changing their approach over the years, extending that definition. To understand how vague and complex the task of defining an outlier is, Ayadi et al. [10] presented twelve distinct interpretations from the point of view of several authors. Nowadays, despite the ambiguity in finding a clear definition, an anomaly or outlier can be defined as "data points that are significantly different from the majority of the data points or which does not conform to the expected notion of normal behavior" [43]. In an implicit way, the assumption made is that most of the data is being generated through a normal process and anomalies are the observations that are being generated by a distinct mechanism.

In order to help solving the problem of finding unexpected events or items in data that does not conform to a well-defined notion of normal behavior, several anomaly detection techniques have emerged. However, selecting a suitable detection technique that meets

the application requirements is a non-trivial task. In fact, this goal faces some challenges: the absence of ground truth for training and/or validation of models used by anomaly detection methods, given that getting them in an accurate and representative way is an overwhelming task; the imprecise boundaries between anomalous and normal behavior; the notion of normal behavior are in constant progress and consequently the current concept cannot be a correct representation in the future; the idea of an anomaly differs from domain to domain making the application of methods developed in one field to another a difficult task; the detection and removal of noise in the data which tends to behave like real anomalies; and the existence of high-dimensional datasets across several fields of study [60].

To address those issues, extensive research on anomaly detection techniques in numerous research areas and application domains has been made. That being said, several surveys providing a structured and comprehensive guideline to anomaly detection techniques of the research on anomaly detection in general were presented [18, 4, 64, 2, 43, 89, 26, 23, 35], allowing the understanding of which one is more effectiveness for each domain of application. However, due the large number of applications areas that anomaly detection covers, it is unfeasible to cover all of them in just a single survey, for the sake of space limitation. Consequently, there are a considerable number of researches for specific data domains such as high-dimensional data [81, 49, 96], temporal data [39], data streams [82], network data [76, 34, 73], uncertain data [3], intrusion detection [74, 29] and novelty detection [66]. In recent years, more modern studies have been published, especially in the area of deep learning [17, 61, 48] and ensemble techniques [54, 62, 90].

Before selecting an applicable anomaly detection method to detect anomalous instances in the application domain under study, understanding the different aspects of an anomaly detection problem is a crucial requirement. The first key aspect is the nature of the input data in terms of identifying the set of attributes (univariate or multivariate), the data type of each one of them (continuous, categorical or binary), the existence of ground truth (labelled or unlabeled) and the size and dimensionality of the input data.

Secondly, it is imperative to have a clear idea of the distinct classifications of anomalies. In most cases, we aim to detect single anomalous instances in data, called *point anomalies*. However, a data instance can be seen as normal in a given context, but when a different context is considered, it may ends up to be an anomaly. This type of anomaly is named a *contextual anomaly* (also referred to as *conditional anomaly* [78]). Furthermore, an individual data point who was not considered an anomaly in any of the types previously presented, may still be recognized as an anomalous point along with a collection of related data instances, called a *collective anomaly*. Note that it is possible to detect collective and contextual anomalies using point anomaly detection algorithms as explained in [35] and in [18].

Finally, it is extremely important to know the existing anomaly detection techniques

as well as the corresponding strengths and weaknesses. Depending on the availability of labels in the input data, we can categorize the anomaly detection methods in three modes: supervised anomaly detection, semi-supervised anomaly detection and unsupervised anomaly detection. Supervised anomaly detection techniques assume the existence of labeled data as input, where every instance are identified as belonging to the normal or anomalous class, applying supervised algorithms to the training dataset. In semi-supervised detection algorithms instances from the normal class are trained, and then the model learns to recognize anomalous instances in the test data. Since unsupervised anomaly detection methods do not require any labelled input data, they are the most useful, challenging and researched techniques in this field.

2.2 Unsupervised Anomaly Detection Algorithms

2.2.1 k-Nearest Neighbors

k-Nearest Neighbor (kNN) algorithm [72] is one of the most commonly used neighbor-based (or distance-based) outlier detection algorithms and assumes that anomalies are data points at a far distance from their closest neighbors. This method assigns a anomaly score to each data point based on its distance to its kth nearest neighbors and then considers a desired number of top ranked outliers to be anomalies.
Although the authors have shown that the partition-based algorithm for mining outliers is highly efficient and that the kNN algorithm outperforms the nested-loop and index-based algorithms, there are several improvements that need to be made leading to the development of several variants, such as an easy mechanism to find a proper distance or similarity measure that scales well with a low computational cost for a large and high-dimensional dataset; a way to find a appropriate value for the distance threshold d and for the parameter k; addressing the issue of quadratic complexity; and allow to find local outliers.

2.2.2 One-Class Support Vector Machines

One-Class Support Vector Machines (One-Class SVM) algorithm [75] is an unsupervised model-based anomaly detection approach extending the support vector machines (SVM) algorithm to the case of unlabeled data. In order to group the input data points, employs SVMs to learn a high-dimensional space decision boundary via a kernel, iteractively performing dot products between the data points to find a *maximal margin hyper-plane which best separates the training data from the origin*.
In this method, overfitting is prevented by allowing a percentage of data points to fall outside of the decision boundary. Besides that, the authors showed that is well-behaved in terms of computational complexity. However, in this “easy-to-use black-box method”

the selection of kernel parameters is not an easy task.

2.2.3 Local Outlier Factor

Local Outlier Factor (LOF) algorithm [15] is one of the most popular density-based outlier detection algorithms. This technique was conceived to find outliers in a multidimensional dataset and to address the challenge of comparing points with the neighborhood in areas with different densities. The LOF algorithm determines the likelihood of each data point being a local outlier assigning it an anomaly score, known as Local Outlier Factor. Assuming that anomalies are significantly farther from its neighbors than they are from each other, the LOF score is given by the ratio of the average distance to its k nearest neighbors and the average distance of each of those neighbors to their k nearest neighbors, such that anomalies will have higher LOF score.
The authors showed that LOF outperforms One-Class SVM when applied on real-world datasets. Besides that, this algorithm requires a single parameter: the number of nearest neighbors used to define the local neighborhood of an instance (k). However, since it is computationally expensive and the choose of a proper value of k is non-trivial task, many extensions of this algorithm have been proposed [83, 46].

2.2.4 Cluster-Based Local Outlier Factor

Cluster-Based Local Outlier Factor (CBLOF) algorithm [42] is a clustering-based technique for detecting local outliers and outlying clusters in the input data, based on the following assumption: *normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters* [18]. This method provides importance to the local data behavior, determining outliers clustering the input data and then using a LOF-based computation with a new anomaly score measure, known as Cluster-Based Local Outlier Factor (CBLOF), corresponding to the degree of each data point being an outlier. The CBLOF score is determined considering the distance of each data point to its cluster centroid multiplied by the size of its cluster, such that the higher the CBLOF of a cluster is, the more anomalous would be.
The authors showed that this approach outperformed the existing techniques, having a linear computational complexity.

2.2.5 Isolation Forest

Isolation Forest (iForest) algorithm [59] is a model-based approach that focuses on separating outliers from the rest of the data points based on the principle that anomalies are rare and therefore more susceptible to isolation. This method builds an ensemble of isolation trees (iTrees) using the random forest technique to compute the isolation score for each data point according to its distance from the root of the tree. The intuition is that

anomalies are instances with shorter average path lengths on the iTrees, that is, are isolated closer to the root of the tree instead of the leaves.
In this algorithm only two parameters need to be set: the number of trees to build and the sub-sampling size. Liu et al. [59] stated that this algorithm has a linear time complexity and attested that performs better than LOF on large and high-dimensional datasets.

2.3 Clustering

Clustering is a popular unsupervised learning method that groups data points based on the similarity and dissimilarity between them such that similar data points are assigned to the same cluster and dissimilar data points are assigned to other cluster.
However, the definition of *similarity* is subjective, depending on what is intended to be achieved in this task. There is a range of Clustering Methods that differ in their assumption, such as Density-Based Methods, Hierarchical-Based Methods, Partitioning-Based (or Centroid-Based) Methods and Grid-based Methods.

2.3.1 k-Means

k-Means is one of the simplest unsupervised learning method which follows a centroid-based approach to the clustering problem. This algorithm works as follows. Once chosen the number of clusters, k, randomly selects the centroid – a data point representing the center of the cluster – for each cluster. After the centroids are initialized, iteratively assigns each data point to the closest centroid and recomputes the positions of the centroids of the new clusters through the mean of all points for each cluster, until the sum of the squared error (SSE) is minimized. However, choosing the right number of clusters is not always an easy task. It requires domain knowledge combined with clustering evaluation metrics such as the elbow method and the silhouette coefficient.
A major limitation of the k-Means is that cluster boundaries must be linear, meaning that the algorithm will generally be ineffective if the clusters have irregular shapes.

2.3.2 Spectral Clustering

Spectral Clustering is a flexible density-based clustering method that is able to find non-linear boundaries between clusters. This algorithm can be seen as kernelized k-means with roots in graph theory since it compute a low-dimensional representation of the data using the spectrum (eigenvalues) of a nearest neighbor's graph – the affinity matrix - , and then assigns labels applying a k-means algorithm (or a "discrete" strategy, alternatively).
As with k-Means, Spectral Clustering requires the number of clusters. Despite having a good performance for a small number of clusters, it is not recommended for many clusters.

2.4 Dimensionality Reduction

2.4.1 Principal Components Analysis

Principal Component Analysis (PCA) is a commonly used dimensionality reduction technique in machine learning [47]. The aim of this technique is to project the input data into a lower dimensional space, without losing relevant information. As such, the feature vector is transformed into a new vector of k orthogonal variables, called Principal Components. Those components are uncorrelated and incorporate the least amount of information regarding to the variance of data input data, therefore the remaining components can be ignored.
Employing this method to the input dataset before the anomaly detection algorithm application, allow us to address the data sparsity and the computational complexity problems. In the end, the result is mapped into the original feature space. Besides that, this dimensional reduction technique can also be used as an anomaly detection technique assuming that deviations from the normal subspaces may represent anomalous points.

2.5 Visualization and Interaction

2.5.1 Parquet

Parquet[1] is a columnar storage file format in the Apache Hadoop ecossytem. This format was developed in order to efficiently store large volumes of data and is used as an alternative to CSV file format when the number of observations in the dataset is high[2].

2.5.2 Kibana

Kibana[3] is an open-source interface for data exploration, visualization and discovery stored in ElasticSearch. In other words, it can be defined as a browser-based search dashboard and analytics for Elasticsearch[4]. An example[5] of an interactive visualization created in Kibana is presented in Figure 2.1.

[1] https://parquet.apache.org/
[2] https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d
[3] https://www.elastic.co/kibana
[4] https://github.com/elastic/kibana
[5] https://www.elastic.co/guide/en/kibana/current/dashboard.html

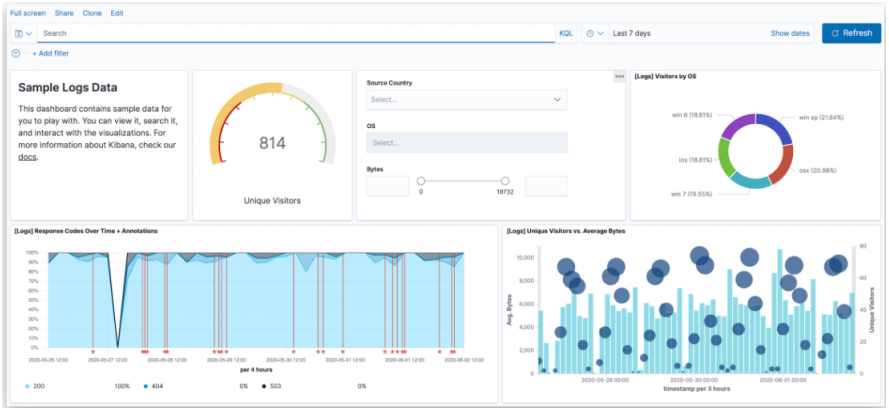


Figure 2.1: Example of a Kibana Dashboard.

2.6 Fundamental Tools and Libraries

2.6.1 Dask

Dask[6] is a Python parallel computing library known for being an alternative to one of the best and most popular data science tools for data cleaning and analysis used in Python – Pandas – when dealing with large or even larger-than-memory datasets. Besides Pandas, integrates other open-source projects as Numpy and Scikit-learn in such a way that Dask Arrays, Dask DataFrames and Dask-ML are the equivalent of Numpy Arrays, Pandas DataFrames and Scikit-learn, respectively. With more detail, Dask stores the complete data structure on the disk, and dividing it into chunks (or partitions) computes functions to those blocks in parallel. Additionally, all intermediate tasks once completed are cleared it to save the memory consumption. Along with that, Dask can either run on a local machine or in a distributed manner across multiple nodes in a local cluster, using Dask's *dask.distributed* Scheduler that only requires the creation of a *Client*.
To address some issues in terms of models and/or datasets dimensionality that may be faced while running most popular Machine Learning libraries, Dask-ML library provides a distributed and parallel way to run them. Some examples of available methods are presented below.

- Sklearn's preprocessing methods such as StandardScalar, RobustScalar, LabelEncoder, and OneHotEncode;
- Sklearn's methods hyperparameter search like GridSearchCV and RandomizedSearchCV;
- Sklearn's Linear Models as LinearRegression and LogisticRegression.
- Only two Sklearn's Clustering Models - Kmeans and SpectralClustering.

[6]https://dask.org/

- Sklearn's optimization algorithms such as admm, lbfgs and gradient_descent.
- Sklean's regularizers like L1, L2 and ElasticNet.

In addition to Dask's performance and scalability advantages, it uses a familiar syntax, where Dask's commands are particularly similar to the ones in Pandas, which requires minimal code changes without having to get a better infrastructure (like Spark). However, not all of its functions are implemented in Dask. Furthermore, since most Dask functions are *lazy*, a compute method needs to be called in order to execute the operation.

2.6.2 PyOD

PyOD[7] is an open-source Python toolbox for performing scalable outlier detection on multivariate data [95]. Created to fill the gap of a suitable Python toolkit for anomaly detection, provides a collection of outlier detection algorithms ranging from supervised learning to unsupervised learning techniques, as well as outliers ensembles and neural network-based approaches, all of them derived from a base class with the same interface as scikit-learn's API. Furthermore, for robustness and scalability optimization employs just-in-time (JIT) compilation and parallelization whenever possible.

2.6.3 Plotly

Plotly[8] is browser-based graphing library available in Python that allows the creation and display of interactive visualizations in Jupyter Notebooks. This visualization tool is indispensable when facing a big data problem, given that sometimes the volume and variety of data is so big that it is impossible to represent information in a single static plot such that all points, lines or bars are visible by a human and there is no overlap between them. Having said that, Plotly provides a wide range of interactive visualization techniques, allowing a more in-depth exploration and understanding of the data.

2.6.4 T-Distributed Stochastic Neighbouring Entities

T-Distributed Stochastic Neighbor Embedding (t-SNE) [88] is a dimensionality reduction technique that allows high-dimensional data to be visualized efficiently through its mapping on a two or three-dimensional space. This technique aims to find the most suitable way to match the pairwise similarities' distribution of the original data with the pairwise similarities' distribution of the corresponding low-dimensional points in the embedding. The authors showed that t-SNE outperforms existing state-of-the-art techniques for visualization. However, its computation could be quite complex and heavy, and may require the previous application of another dimensionality reduction method.

[7] https://pyod.readthedocs.io/
[8] https://plotly.com/

Chapter 3

Related Work

Anomalous User Behavior Detection has become the central focus of several information security systems, such as insider threat detection, intrusion detection and authentication systems. Haystack [85] presented a comprehensive crowd-based research showing the growing number of internal attacks in organizations. The inside attacks made by attackers that already have credentialed access to an organization's networks and/or services proved to be significantly more damaging and are more challenging to detect than outside attacks. As stated in the 2010 SANS Institute report [45], in the United States alone, internal attacks cost around $400 billion per year, where $348 billion are associated to privileged users. However, it is crucial to be able to distinguish between malicious insider attacks and anomalous activities without malicious intent, such as errors or human mistakes. Besides that, the malicious user behavior may be associated to a stolen login credentials circumstance.

Given that investigating attacks by analysts is a complex, costly and time-consuming task, as they have to deal with millions of raw system logs, a creation of an automated forewarning system is an imperative requirement in this context. To solve this problem, several anomaly detection techniques based on machine learning have emerged. With a statistical analysis of user's actions and the application of those techniques over that information, it is possible to profile users by understanding the user's behavior tendencies and preferences. Any pattern, event or observation that deviates from those expected behaviors is treated as an anomalous user behavior and, consequently, a potential attack. Whenever one of those deviations occurs, an alert to the system administrator is raised in order to determine the probability of being a harmful attack before it produces negative effects on the organization.

Due the relevance of this topic, significant researches have been provided to find a suitable way to prevent and detect those attacks. In the early 21st century, primordial studies on unsupervised learning for anomaly user behavior detection were presented in [28] and in [58]. After that, several researches focused on the computer system logs to build a user profile using neighborhood-based approaches [79], statistical models [38, 63, 20], collab-

orative approaches [21, 50] were published.
Lazarevic et al. [53] produced a comparative evolution of intrusion detection algorithms such as k-Nearest Neighbors (kNN), one-class Support Vector Machines (SVM), Local Outlier Factor (LOF) and Mahalanobis-based. Likewise, Eskin [27] compared three unsupervised anomaly detection algorithms for intrusion detection: as k-Nearest Neighbors (kNN), one-class Support Vector Machines (SVM) and clustering-based estimation. A comparison of a supervised approach with an unsupervised approach using the Isolation Forest method in order to detect insider threat from network logs was presented in [31]. Shashanka et al. [77] proposed a User and Entity Behavior Analytics (UEBA) module of the Niara Security Analytics Platform to detect anomalies in users accessing servers within an enterprise through the application of the SVD-based algorithm. Li at al. [57] implemented a security audit technology to detect the anomalous behavior of database operations applying the one-class Support Vector Machines algorithm. Sun et al. [80] have proved the efficiency of the Isolation Forest algorithm for detecting anomalous user behavior building an automatic anomaly user detection system which uses an extend version of these algorithm to support categorical data dimensions and evaluating that framework using many log files belonging to a real enterprise dataset. A comparative survey on nineteen anomaly detection algorithms for multivariate data were provided in [35]. In [41], six supervised and unsupervised algorithms for system log analysis were confronted. Lashkari et al. [52] provided a comprehensive overview of previous user profiling researches and methods, presenting their strengths and weaknesses, and, consequently, proposed a user profiling model that covers all available sources and related features based on the cybersecurity context. Xi et al. [90] introduced an ensemble algorithm that combined one-class Support Vector Machines, Recurrent Neural Network and Isolation Forest algorithms that showed a great performance in detecting user's anomalous accesses within enterprise context. A comparing anomaly-based for Intrusion Detection Systems baseline, selecting twelve algorithms from six families of unsupervised algorithms, were developed in [29], finding that algorithms belonging to the classification family (such as one-class SVM and Isolation Forest algorithms) tend to show better scores.
However, the dataset used for user behavioral analysis is usually large in dimensionality and in size and most of the conventional algorithms, such as statistical-, nearest neighbour- and clustering-based techniques, tend to have more difficulty detecting anomalous behavior and, because of that, a perfect trade-off between the rate of detection and the time complexity of those higher dimension datasets is hard to get. Because of this, Prarthana et al. [70] developed one of the first user behavior anomaly detection systems for detecting anomalies in multi-dimensional event log data by analyzing the statistical behavioral patterns. Besides that, more sophisticated researches using unsupervised deep learning approaches to detect anomalous activity from system logs in real time [87, 13]

were developed. Given its complexity and for lack of time, this topic will not be covered in this work.

Moreover, a previous project also focused on detecting anomalous user behavior but with a different purpose was performed in the company [67]. It was the first approach on anomaly detection through machine learning techniques problem and it dealt with user accesses to clients' telephone number information. The aim of that project was to detect illicit accesses to that information, focusing on those who are most likely to match information theft actions. The project was structured as follows. First, it was decided which features should be created or extracted from the access logs to build the datasets where the anomaly detection algorithms will be applied. After that, the required preprocessing and normalization procedures were executed, creating two datasets - one dataset in order to users and another in order to telephone numbers, where the features of the users dataset focused in describing each user behavior in terms of the number of times a user made accesses, to how many telephones and in how many days. An analogous approach was considered for the telephone numbers dataset. After those steps, all the requirements so that they could detect anomalies in the datasets were met. Thus, the algorithms considered were clustering methods such as k-Means, DBSCAN, and Affinity Propagation; and two anomaly detection methods, Elliptic Envelope and Isolation Forest. To determine optimal parameters for those algorithms, parameter ranges were defined and score tables were created with the results obtained from different combinations of parameters. To obtain specific results for different analytic perspectives, besides being applied on the entire datasets built, the algorithms were also applied to different combinations of some of their features. Therefore, ensemble methods were chosen to ensemble the various results. Finally, decision rules were created to classify the anomalies in different anomaly classes, such as each anomaly detected was associated to a class that categorized its behavior. However, this project focused on detecting a set of known anomalies related to the volume of accesses.

Despite several studies devoted to anomalous user behavior detection within an organization, due to variability in the application context, where some organizational-dependant components may need to be taking into account, and to the lack of research focusing on a large real-world organization datasets, perhaps for confidentiality, it is still unclear which are the more suitable detection approaches and algorithms for this purpose.

Table 3.1 presents a comparative table with a short summary of the approaches adopted in related works with the same aim. The methodology adopted in this project is based on the work developed in *Detecting Insider Threats Using RADISH: A System for Real-Time Anomaly Detection in Heterogeneous Data Streams* [14], in *Detecting Insider Threats in a Real Corporate Database of Computer Usage Activity* [86], and in *Automated Insider Threats Detection System Using User and Role-Based Profile Assessment* [92].

Table 3.1: Related work summary

Name	Dataset	Feature Extraction	Contextual Information	Anomaly Detection	Evaluation	Limitations
[51]	Synthetic insider threat test datasets provided by CERT.	User activity modeling based on daily activity summaries.	Each user's behaviors are described by: user activity logs, user's e-mail content, user's e-mail account and sending/receiving information.	Four one-class classification algorithms and their combinations.	-	Dependent on ground truth and insider threat scenarios made available by each company. Not dependent on peer group activity.
[40]	Synthetic insider threat test datasets provided by CERT.	Daily summary of each employee's activity.	-	Meta-classifier of multiple classifiers: Neural Network, Naive Bayesian Network, Support Vector Machine, Random Forest, Decision Tree, and Logistic Regression.	Ground truth covering distinct insider threat scenarios provided by an expert Red Team.	Dependent on ground truth and insider threat scenarios made available by each company. Not dependent on peer group activity.
[86]	Collection of synthetic insider threat test datasets provided by CERT.	Semantic and structural features in seven categories and a range of feature normalizations.	Normalizing each day's value to: all the user's days to highlight unusual days; all users on that same day to dampen day-to-day, across-the-board variations; a user's group to heighten variation from peers.	Multiple Anomaly Detection algorithms are applied and evaluated.	Ground truth covering distinct Insider Threat scenarios provided by an expert Red Team.	Lack of evaluation methods for anomaly detection algorithms.

Continued on the next page

Table 3.1: Related work summary (cont.).

Name	Dataset	Feature Extraction	Contextual Information	Anomaly Detection	Evaluation	Limitations
[56]	Synthetic insider threat test datasets provided by CERT.	User activity modeling based on daily activity summaries.	User's daily observations, comparisons between the user's daily activity and their previous activity, and comparisons between the user's daily activity and the previous activity of their role.	PCA decompositions for each anomaly metric.	Ground truth covering distinct insider threat scenarios provided by an expert Red Team.	Does not take advantage of Machine Learning-based algorithms for Anomaly Detection.
[92]	Synthetic insider threat test datasets provided by CERT.	User activity modeling based on daily activity summaries.	Considers a baseline population for comparison (a peer group), a time period for the baseline activity (user's own past activity), a time granularity for potential detection (day).	Unsupervised Ensemble-based Anomaly Detection technique.	Ground truth covering distinct insider threat scenarios provided by an expert Red Team.	Lack of ground truth and evaluation methods for Anomaly Detection algorithms.
[55]	Collection of synthetic insider threat test datasets provided by CERT.	Frequency features and statistical features exploring different levels of granularity.	User-user relationships, user-PC relationships, regular work hours, and website and file categories.	Three supervised learning algorithms: Artificial Neural Network, Random Forest, and Logistic Regression.	Ground truth covering distinct insider threat scenarios provided by an expert Red Team.	Dependent on ground truth and insider threat scenarios made available by each company. Not dependent on peer group activity.

Continued on the next page

Table 3.1: Related work summary (cont.).

Name	Dataset	Feature Extraction	Contextual Information	Anomaly Detection	Evaluation	Limitations
[25]	Real-world dataset provided by a large defense contractor.	Daily user statistics across six distinct activity domains	Peer group inconsistency across domains and user inconsistency across time.	Ranking based fusion scheme combining a Markov Model approach with a supervised learning approach.	Synthetically injected anomalies based on real-world malicious behavior scenarios.	Dependent on ground truth and insider threat scenarios made available by each company. Not dependent on peer group activity.
[44]	Two real-world datasets from large enterprises.	User activity modeling based on daily traffic frequency counts of meta-events from multiple sources.	Comparisons with the user's own past activity.	PCA-based user activity behavior Anomaly Detection method.	Qualitative evaluation by security analysts examining raised alerts.	Not dependent on peer group activity.
[32]	Real-world dataset named 'Vegas'.	Privacy preserving feature extraction approach of daily user online activity.	Organizational hierarchy.	Isolation Forest Anomaly Detection algorithm.	Artificially injected insider threat events via a collection of Red Team users.	Not dependent on peer group activity.
[14]	Real-world dataset named 'Vegas'.	Average, weighted average, and average difference of user session's activity summaries from all domains.	Comparisons with the user's own history and with their peers.	Modified version of the Isolation Forest algorithm that identifies the anomalous points and the corresponding features primarily responsible for anomalies.	Artificially injected insider threat events via a collection of Red Team users.	-

Chapter 4

Analysis

This dissertation was carried out as part of a wider interdisciplinary project under development in the Cybersecurity Department (DCY) of Altice Portugal with the support of experts in the fields of Data Science, Computer Science and Cybersecurity. The aim of this project is to develop and integrate a robust automated system that is capable of controlling and mitigating malicious activities by users with authorized access to the company's systems in order to avoid or minimize damage to the company. More specifically, based on near real-time analysis of user activity logs, the system identifies anomalous user behavior and triggers an alert if the risk of being an insider threat is extremely high. In fact, the key requirements for an effective detection system are the following:

R1) Automatically identify the users with a high risk of being a potential threat while they are active in the company's system.

R2) A robust and flexible system in continuous learning capable of adapting to the dynamic nature of users and entities activities.

R3) The system should be also able to detect distinct types of known insider threats, such as information system sabotage (for instance, when a user impairs the quality of a service to cause loss of confidence on part of its customer), intellectual property theft (for example, when the intellectual property rights like copyrights or patents are violated by a user), and fraud (information and property extortion), as well as unknown cases of insider threat.

R4) The detected anomalous user behavior must take into account the local context, considering the user's long term historical profile along with that user's peer group long term historical profile.

R4.1) The system should be able to identify the type of account that performed the action and consider its nature in the anomaly detection process. In this project, these accounts are expected to behave quite differently differing in terms of

access volume, time, location, and permissions. An account can be one of the following three types:

* Application account - Accounts used by applications and services to access databases, run scripts, or provide access to other applications.
* Database Administrator (DBA) account - Accounts belonging to privileged users with administrative access (highest privilege) to one or more systems.
* Nominal account - Personal accounts with moderate privilege to access and query databases. Mainly, these user accounts are not allowed to make changes in the system.

R4.2) Let us suppose that one day users with similar roles and, consequently, with a similar account's past behavior, have a volume of accesses much higher than what was expected. In this case, if you examine the behavior of each account individually, it is reasonable to assume that it is anomalous with respect to its past behavior. However, if the behavior of the other accounts with similar past behavior were also analyzed, these are no longer considered anomalies. Having said that, the system must consider three types of anomalous behaviors: anomalous behavior with respect to another accounts, anomalous behavior with respect to the group of similar accounts, and anomalous behavior with respect to past activity.

R5) The flagged alerts must be self-explanatory, allowing security analysts to easily locate and visualize detected insider threats.

Some rule-based approaches built by domain experts have been adopted to detect insiders' malicious activities, however those approaches prove to be infeasible since they do not fulfill the requirements previously presented.
The aim of this dissertation is to address this limitation by developing an approach that combines the knowledge of experts with the application of machine learning-based anomaly detection algorithms. A machine learning-based strategy automatically identifies anomalous user behavior without prior knowledge or rules, allowing a constant learning and updating of the anomaly detection algorithms. This machine learning-based approach focuses not only on detecting account behaviors that are persistently abnormal (anomalous user accounts), but essentially on detecting deviations in user accounts behavior from their own or a specific baseline (anomalies).
To achieve this goal, an Exploratory Data Analysis and Feature Selection Module, a Data Extraction, Preprocessing and Preparation Pipeline, a User Accounts Characterization and Anomalous User Accounts Detection Module, an Anomaly Detection Component, and an Anomaly Characterization Module were developed.

4.1 Data Description

The dataset used in this project refers to more than two billion user access log records from over four thousand distinct users belonging to a real enterprise, where each log record typifies a user access to a specific database. In fact, to better understand the scale of the dataset, about 7.8 million user access log records are generated per day. This dataset relates to a range of time of six months – from January 2019 through June 2019. Each record contains information about the timestamp, a user ID, an account ID, a device ID, an object ID, and an activity name (e.g., login and read), interleaved with some user-, account- and object-metadata to provide detailed context.
Those collected raw user access log records were stored and enriched with metadata in ElasticSearch. However, as aimed in this work, to preprocess them and detect anomalous users' accesses we have extracted them to a Python Jupyter Notebook. Table 4.1 presents a brief description for each of the 38 features.
As earlier mentioned, in anomaly detection process, big data represents a massive computationally challenging task that is not directly proportional to the overall performance, specifically regarding to the detection rate. Besides that, dealing with multivariate categorical features is another obstacle in this context regarding that several values of a categorical attribute cannot be ordered, also contributing to the computational complexity of the model and harming the performance.

Table 4.1: Data Description

Field Name	**Field Description**
action	Action being executed. The usual values are *login*, *logout* and *read* but this work only covers login and read actions.
action_details	If the action is *read*, indicates the command executed, e.g., *select*, *update*, *drop*. Otherwise, the value is *null*.
action_result	If the action was successful or not. Although the action could not be done for some reason, for example an error with the password, in our dataset we are only dealing with successful actions. For the same reasons presented in the previous feature description, we are also going to ignore this one.
actor	Account name that executed the action.
actor_account_risk	Account permissions level.
actor_account_state	Account state (active or not active).

Continued on the next page

Table 4.1: Data Description (cont.).

Field Name	Field Description
actor_company	Company to which the account belongs. Making a simple analysis on the distinct values in this field, it was possible to observe that most of the instances have an unknown or null value associated. For that reason, this field was ignored.
actor_department	Department to which the account belongs. Given the non-significant number of records with a non-unknown or non-null value in this field, this was ignored.
actor_details	Details about the account.
actor_domain	Domain to which the account belongs.
actor_executable	Program used to know where the access was made.
actor_identity_id	User ID, i.e, the identifier of the owner of the account that executed the action. Here we are ignoring a possible password steal from another user. This field is important for our analysis since each user could have more than one account, one per system for example.
actor_identity_risk	User permissions level.
actor_ip	IP from where the access was made.
actor_ip_as_country	Country associated to the IP from where the access was made.
actor_ip_as_name	Autonomous System name of the IP from where the access was made.
actor_ip_as_number	Autonomous System number of IP from where the access was made.
actor_ip_hostname	Hostname, name of the machine from where the access was made, resulting from an IP translation, when possible.
actor_ip_range_type	If client IP is *static* or *dynamic*.
actor_location	Detailed location from where the access was made.
actor_location_geo	Geospatial coordinates from where the access was made.
actor_location_id	Identifier of the location from where the access was made.
actor_network_protocol	Protocol of the network from which the access was made.
actor_network_range	Range of the network from which the access was made.
actor_network_type	Type of the network from which the access was made.
actor_range_type	To distinguish whether the access was made from a data center (servers where the application runs) or a local machine.
actor_type	If it is proper to associate the actor to a person or not.

Continued on the next page

Table 4.1: Data Description (cont.).

Field Name	Field Description
collector	Machine that collected the information.
count	Number of events. Since the value in this field was the same for all records (*1*), this field was not considered relevant for the analysis.
db_brand	Accessed Database Management System (DBMS).
decorated	Engines used for data enrichment.
object	Database identifier with the following format: [*IP*]:[*instance*]:[*database*].
object_group	Application group which the database belongs.
object_ip	Accessed Database IP.
object_type	As expected, due to our interest in analysing the accesses to databases, the value in this field is *bd* for all instances. Due that, this field was ignored in our analysis.
source	Event source. As mentioned before, a filtering in this field (*dams-login*) was applied. Given that, this field is not interesting for the analysis.
technology	Technology used to collect the information.
ts	Event timestamp.

Chapter 5

Architecture

The architecture of the anomaly detection system implemented in this project is detailed in Figure 5.1. As already mentioned, the architecture of this project is composed of several modules, dependent on each other. A brief description of each of these modules is presented below.

- **Exploratory Data Analysis and Feature Selection Module**: The application of powerful visualization tools, provided by Kibana, combined with the help of Altice's security experts, allowed selecting a target data for investigation as well as discovering useful attributes that can contribute to achieve better results.
- **Data Integration, Preprocessing and Preparation Pipeline**: As already mentioned, the data is stored in ElasticSearch and, in order to develop this project, it is crucial to extract them to a Python Jupyter Notebook. Due to the large volume and variety of event data that needs to be extracted and preprocessed, a distributed processing module was integrated taking advantage of Dask features. This component iteratively extracts the data, cleans it, aggregates it and integrates it in dataset with user account's daily activity summary.
- **Data Transformation Module**: Once all daily records have been processed and integrated, this component computes multiple aggregation and ratio features building a comprehensive characterization of the profile of each user account. Additionally, after user accounts characterization, it also builds a user account groups profiling. Finally, a daily scaled dataset is created taking into account the history of all user accounts, the history of the user account itself and the history of the user accounts group to which it belongs.
- **User Accounts Characterization and Anomalous User Accounts Detection Module**: By employing clustering-based techniques on user account's behaviour profiles it is possible to arrange it into groups of user accounts with similar behavior. Subsequently user account behaviors that are persistently abnormal are detected by identifying a significant deviation compared against those in the same account group.

- **Anomaly Detection Module**: Once the feature set for user accounts' daily observations has been computed, applies and compares the performance of multiple anomaly detection algorithms, after finding the best parameter combinations for each of them. These algorithms output a rank or a score that, according to an established threshold, allows it to be classified as anomaly or not. The performance of the algorithms and the chosen threshold is tested through the injection of ten simulated insider threat scenarios, designed with the help of domain experts.

- **Anomaly Characterization Module**: Once anomalies are detected, it is essential to distinguish them, not only to identify whether it is an illicit access or not, but also to find what type of activity originated it. This characterization makes it easier for analysts to visualize the anomaly, as well as help to identify its risk. Several anomaly metrics have been conceived for this purpose, allowing the visualization of the anomalies detected in a dashboard created in Kibana.

After these components are completed, it is expected that each anomaly, depending on its degree of risk, will be properly addressed, with a brief report attached.

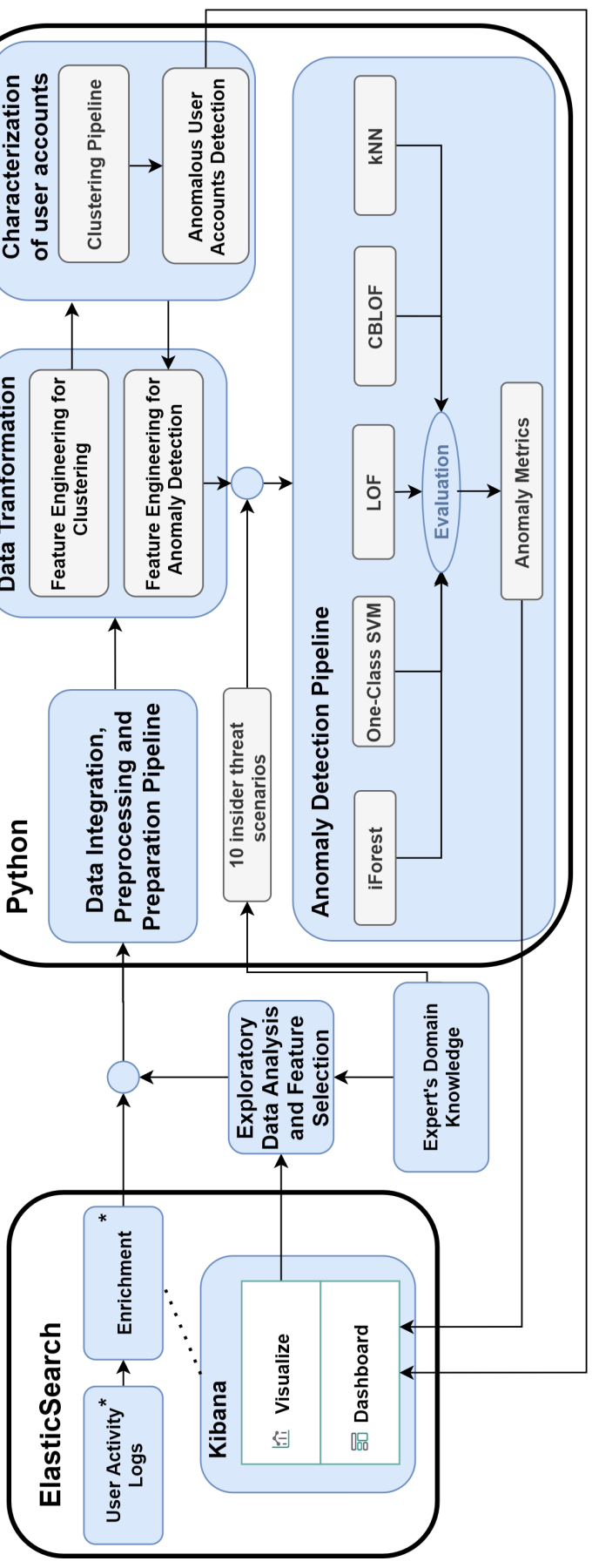


Figure 5.1: Architecture of the Anomaly Detection System. The squares that contain * refer to components that are not covered by this project.

Chapter 6

Implementation

In this chapter, the implementation of the architecture presented in Chapter 5 is described in detail. It will be demonstrated how each of the main components of the system is implemented and how they connect to each other.
The detection system developed in this dissertation was implemented in Python, through the creation of several Jupyter Notebooks, employing available libraries, such as *elasticsearch*, *fastparquet*, *dask*, *numpy*, *pandas*, *scikit-learn*, *plotly*, and *pyod*. In addition, visualization elements were also developed to analyze data and detected anomalies, taking advantage of a powerful visualization tool, Kibana.

6.1 Exploratory Data Analysis and Feature Selection

In order to start the project it is crucial to have a well-defined target of investigation, as well as a subset of relevant features to the problem. This is a fundamental task and will allow a good performance to be achieved.
A study of all dataset columns was performed to gain more knowledge about the data, allowing the discovery of information that was not evident and that deserved to be investigated. Using Kibana, it was possible to get a brief description of the dataset, a summary statistics and distribution of each feature, and to understand interactions between different features. By presenting these analyses to Altice's domain experts, it was possible to choose the relevant features. The set of features chosen for our project must answer the questions presented in the Figure 6.1. Explicitly, in order to guarantee the best results for the project, it is essential to select the fewest features such that they answer all the following questions:

Q1) Who did the access?

Q2) When was the access made?

Q3) How was the access made?

Q4) Where was the access made from?

Q5) What was the access made to?

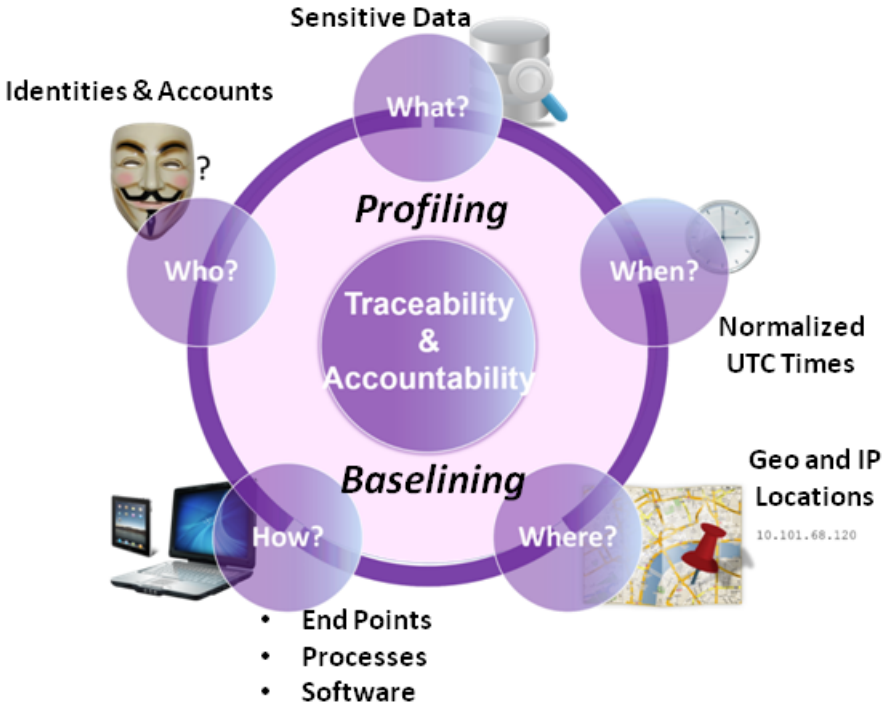


Figure 6.1: Figure made available by Altice's domain experts illustrating the questions that our features must answer.

To answer the first question, both the user identifier (*actor_identity_id*) and the user account identifier (*actor*) can be used. However, since the detection of anomalies will be centered on who made the access, it is decisive to reflect on this choice.
As shown in Figure 6.2, a user (*actor_identity_id*) can have more than one account (*actor*), with different privileges and authorizations assigned to each one. In addition, these accounts can behave quite differently, whether in terms of access volume, time, location, and permissions. Although in many situations service accounts and human user accounts are easily identified by their behavior, they are not classified according to their nature in the dataset. However, it is expected that an account can be one of the following three types: Application account, Database Administrator (DBA) account or Nominal account. For confidentiality reasons, their characterizations will not be presented.
Given that it is clear that the behavior of an account depends on its function, this approach will be centered on each user account (*actor*) instead of the user (*actor_identity_id*).

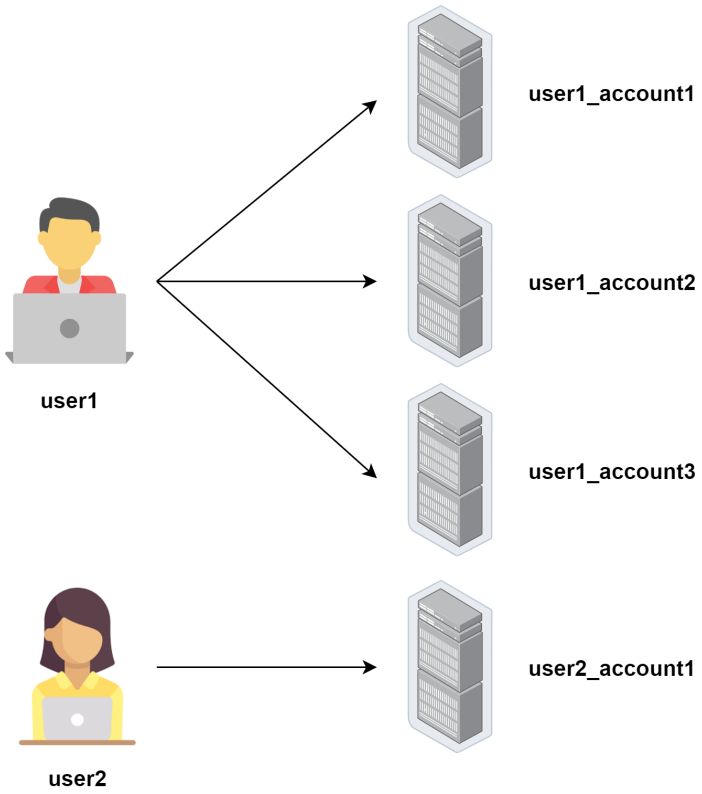


Figure 6.2: Representative example of the relationship between a user (*actor_identity_id*) and a user account (*actor*), showing that a user can own one or more accounts.

Once the first question has been answered, the answers to the next three are easily obtained. As for 'When?', since it is the only field that contains this information, the *ts* is chosen. As to 'How?', although there is no field with this specification, combining the features *actor_range_type*, *actor_executable* and *action_details* provide good insights into this information. To answer 'Where?', the feature *actor_location* was selected.
Finally, it is required to answer the question 'What?'. As expected, knowing that this project focuses on accesses to databases, there are multiple levels of data granularity related to this. Thus, it is crucial to choose a level that provides a good balance between the object's granularity and computational complexity. Those different granularity levels on the accessed object can be found in Figure 6.3.
According to Altice's security experts, the choice of the highest granularity does not affect the overall performance of the system. For this reason, the database management system whose object accessed concerns, expressed by the feature *db_brand*, is chosen to answer the question 'What?'.
Furthermore, a short list of application ecosystems whose in-depth study may be relevant to the business (from the point of view of detecting relevant anomalies in terms of potential illegality) was also provided by the domain specialists, as well as the corresponding filters to be applied on the set of events. Due to confidentiality restrictions, that list will not be shown.

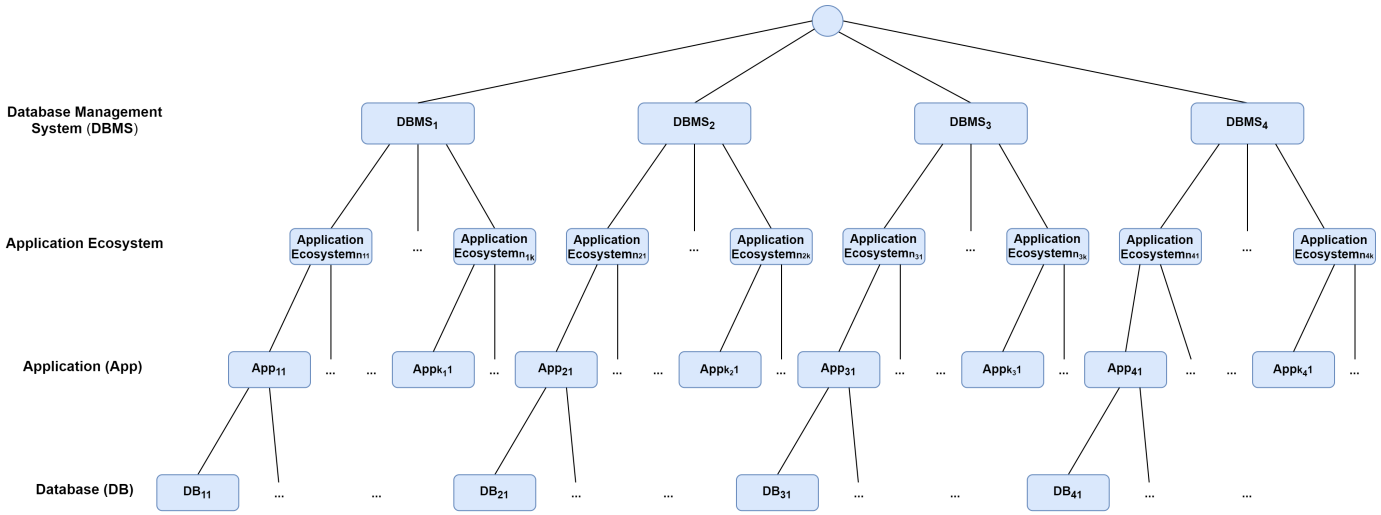


Figure 6.3: Ordered levels of granularity characterizing the accessed object. Note that for confidentiality reasons the values at each level have been obfuscated.

6.2 Data Integration, Preprocessing and Preparation Pipeline

Once the main instances and features are selected, it is essential to load them into Python and preprocess them. Given the volume and variety of the dataset and for reasons of available memory, it is impossible to load it all at once. As such, it was decisive to find an alternative that allowed it to be loaded, processed and reduced in size. Therefore, a distributed processing approach was created and can be found in Figure 6.4. However, note that despite being the same process as the *login*, *read*, *actor_range_type* and *actor_executable* features, the aggregation of the *actor_location* is only done after the next step (*Unstack during_work*), but for reasons of simplification of the schematization as well as its explanation was placed together with the others.
For each day, the system iteratively requests a set of records per hour until all events for the current day have been imported to Python. Once the data is stored on a Dask DataFrame, the data cleaning process for each feature can be started. The entire cleaning process performed for each feature is described below.

1. **actor**

   - Missing Values Treatment: The access log record was removed from the data.
   - Outlier Detection and Treatment: Considering the volume of accesses per user account, outliers were detected using a box-plot for visualization and a rule that determines accounts out of range of -1.5 x IQR to 1.5 x IQR. Once the list of accounts is obtained, their values are analyzed in order to identify their cause. Account records whose *actor* value is a hash or symbol, corresponding

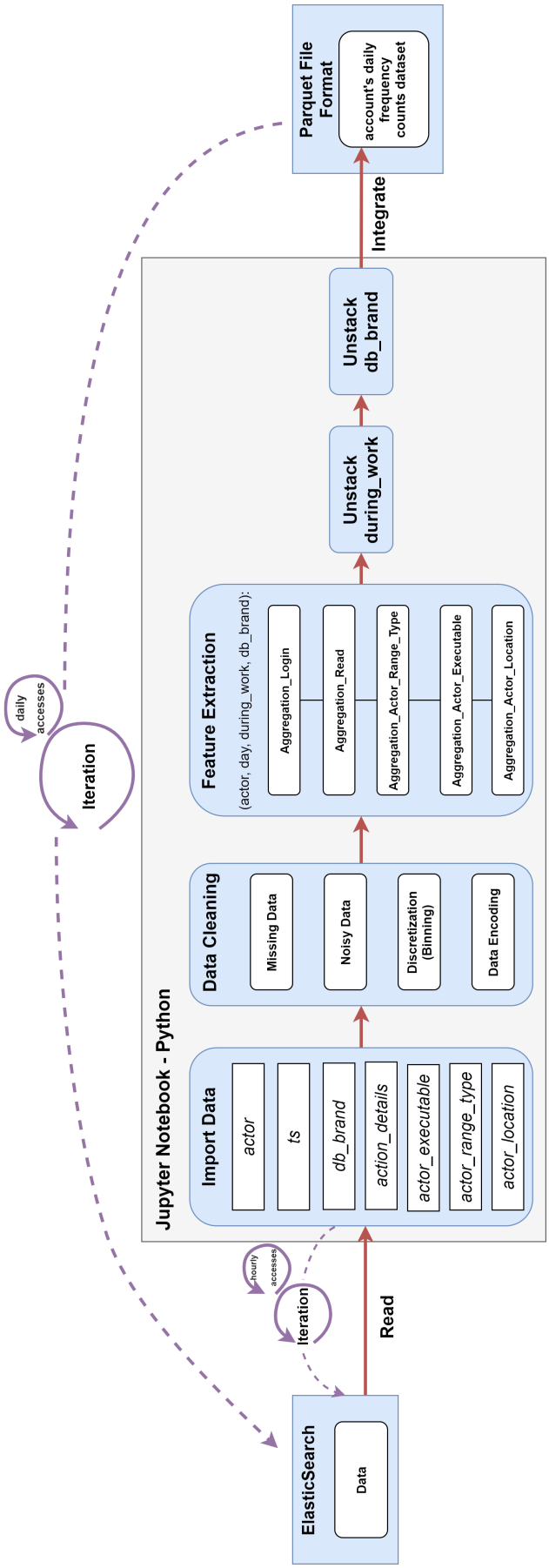


Figure 6.4: Data Pre-processing Pipeline Diagram.

to an extraction or preprocessing error caused by the ElasticSearch system, are removed. In case of a natural outlier it is maintained.

2. **ts**

   - Missing Values Treatment: No missing values.
   - Discretization (Binning): Each value was converted to the corresponding day of the year. In addition, a binary feature was created that indicates whether this access was made during working hours - weekdays between 8 am and 6 pm - or not.

3. **action_details**

   - Missing Values Treatment: As seen in Figure 7.4a, missing values in *actor_details* relate to *login* actions. Thus, *null* values have been replaced by the *login* value.
   - Discretization (Binning): *login* and seven types of most relevant *read* actions were selected. Given that, distinct values that refer to one of these actions have been standardized, otherwise, the value *Other* was assigned.
   - One-Hot Encoding: Converted to nine binary columns (*login, read* and more seven for each read action).

4. **actor_location**

   - Missing Values Treatment: This feature will provide information about the number of distinct locations from which a user account has accessed in a given time range. If an access has no location, there is no way to infer it. As such, this null-value is replaced by *unknown*.
   - Discretization (Binning): Distinct values that refer to the same location have been standardized and the less frequent value were grouped into a single value, *Other*.

5. **actor_executable**

   - Missing Values Treatment: For the same reason presented above, the null values will be replaced by *unknown*.
   - Discretization (Binning): There are thousands of distinct values in this feature. However, when those values were ordered alphabetically it was possible to find that many refer to the same. Their difference is often caused by, for example, a different windows or linux path, a different version or year, or a notice between parentheses or hyphen. Other similarities were discovered using Levenshtein's distance between two words. After that, several rules

1	Row Labels	dtype3	dtype1	dtype2	dtype1+2	p1	p2	Grand Total
43	actor_executable42	44	1794	8538	10332	100%	0%	10376
44	actor_executable43	270	54954	2053	57007	100%	0%	57277
45	actor_executable44	919	7203	25890	33093	97%	3%	34012
46	actor_executable45	23	410	17	427	95%	5%	450
47	actor_executable46	56	21	1013	1034	95%	5%	1090
48	actor_executable47	69	645	33	678	91%	9%	747
49	actor_executable48	189490	1287331	544766	1832097	91%	9%	2021587
50	actor_executable49	2	1	17	18	90%	10%	20
51	actor_executable50	57	302		302	84%	16%	359
52	actor_executable51	24104	114643	1930	116573	83%	17%	140677
53	actor_executable52	67	9	232	241	78%	22%	308
54	actor_executable53	338	10	1019	1029	75%	25%	1367
55	actor_executable54	632	923	299	1222	66%	34%	1854
56	actor_executable55	11	21		21	66%	34%	32
57	actor_executable56	369154	687156	1011	688167	65%	35%	1057321
58	actor_executable57	775	1296	38	1334	63%	37%	2109
59	actor_executable58	2		3	3	60%	40%	5
60	actor_executable59	19		26	26	58%	42%	45
61	actor_executable60	3083	1292	1850	3142	50%	50%	6225

Figure 6.5: Two-way pivot table between *actor_executable* (rows) and *actor_range_type* (columns) values with additional metrics columns.

were created, using regular expressions, to standardize these values. Then, a pivot table was created in excel to determine for each non-null value in *actor_executable*, the number of accesses (sum of count) in each of the values in *actor_range_type* - *dtype1, dtype2* and *dtype3*. For each row, the following two percentages were determined: percentage of *dtype1* or *dtype2* accesses (p1) and percentage of *dtype3* accesses (p2). Finally, the rows were ordered by the first percentage and a separation threshold was chosen such that values with a percentage equal or above it are classified as *exec1*, otherwise as *exec2*. Figure 6.5 shows a subset of rows in the pivot table, as well as the chosen threshold value.

- One-Hot Encoding: Converted to two binary columns (*exec1* and *exec2*).

6. **actor_range_type**

   - Missing Value Treatment: Missing values have been replaced by *unknown*.
   - One-Hot Encoding: Converted to three binary columns (*dtype1, dtype2* and *dtype3*).

In addition, Table 7.5 reveals the number of distinct values for each feature before and after this cleaning process is performed.
Due to the large volume of event data logging user activities that are being considered in this project, it is essential to find an approach to reduce computational complexity of training without degrading performance. Since the dataset has a time series nature, the chosen

approach was to bin events from each user account considering a specific time-frame granularity, such as hour, day, week or month. Once again, the granularity trade-offs have to be considered. Specifically, it is decisive to choose a level of time granularity for potential detection without drastically affecting the computational complexity of the system. With the help of Altice's experts, daily time intervals were chosen for feature extraction. Therefore, for each *(user, day, during_work, db_brand)* quadruple, aggregated features of activity counts were computed as shown in Figure 6.6.

Considering the multi-level index *(actor, day, db_brand)*, the (two) values in feature *during_work* are unstacked turning them into columns, leading to duplication of the feature set size. Thus, the size of the data can be reduced again by aggregating (*.sum()*) the values of the frequency features, for each *(actor, day, db_brand)* triple. Finally, the previous task is equivalently repeated for feature *db_brand*.

Once the current day data is preprocessed and prepared, it is integrated into a final user account's daily activity dataset, stored in a parquet file.

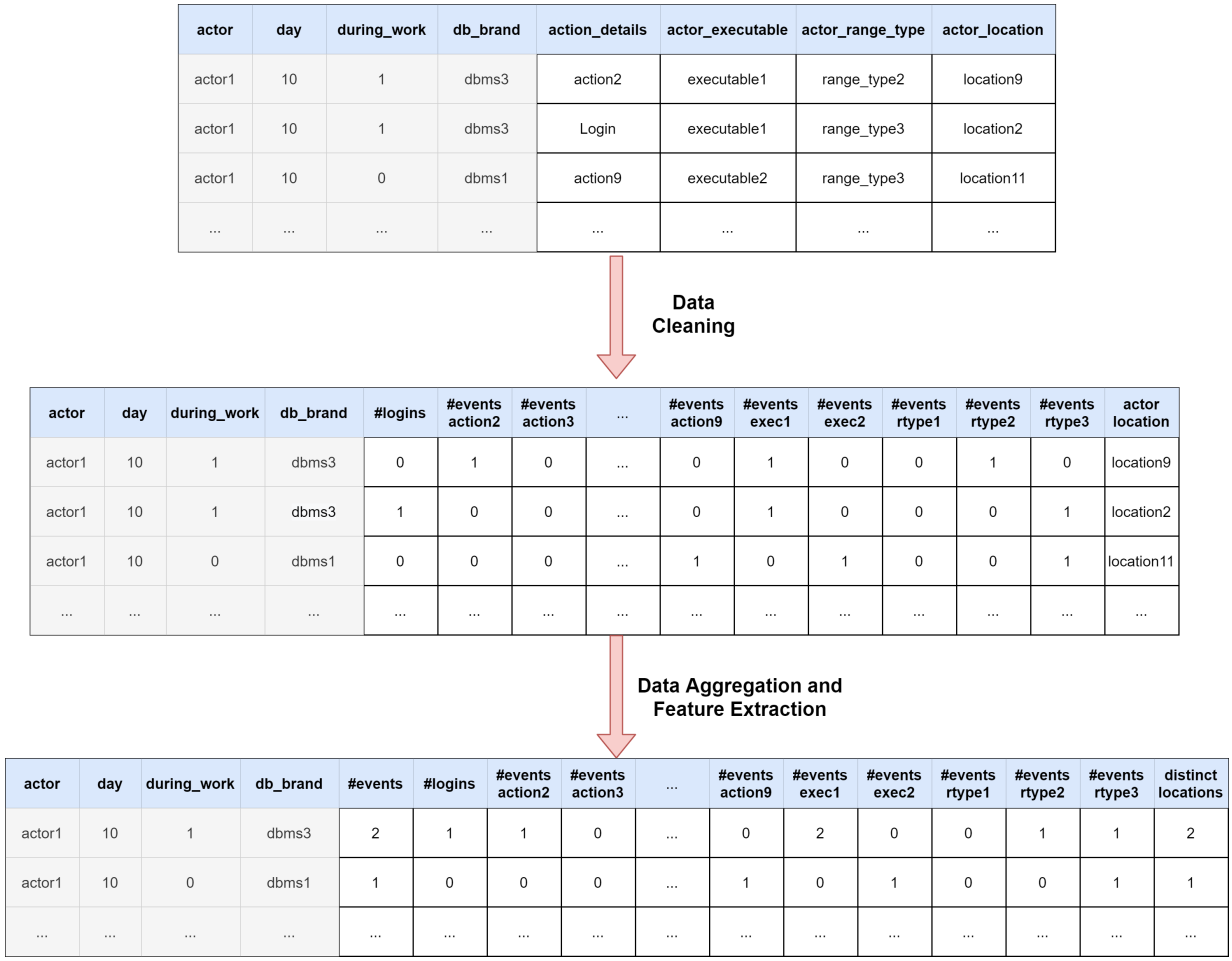


Figure 6.6: Illustrative example of the results obtained after the data cleaning and the data aggregation and feature extraction processes. Due to confidentiality restrictions, the values of each feature were obfuscated.

6.3 Data Transformation

As mentioned earlier, the system was developed considering three types of anomalous behaviors: anomalous behavior with respect to another accounts, anomalous behavior with respect to the group of similar accounts, and anomalous behavior with respect to past activity. However, the latter two types gained more prominence in this investigation. Since there is no information on similar account groups, it is crucial to start by determining them.

6.3.1 Feature Engineering for User Accounts Characterization

Before attempting to group user accounts, it is imperative to build a dataset that describes the overall behavior of each one. Therefore, the dataset was aggregated by user account and metrics summarizing feature's daily behavior were created as follows:

- sum
- mean
- standard deviation (*std*)
- minimum (*min*)
- first quartile (*q25*)
- median
- third quartile (*q75*)
- maximum (*max*)

Regarding the index *day* when aggregating, the distinct number of access days for each user account is counted (*#distinct_days*).
To make it clearer, Figure 6.7 presents an example of the result obtained after aggregating instances with daily frequencies for a specific user account concerning the index *day* and one of the features, *#logins_during_dbms1*.
Note that not all user accounts are interesting, particularly those with few distinct days with accesses. Such accounts were excluded using the following heuristic: if a user account has a number of distinct days with accesses below a certain threshold – 33% of the account's distinct days with accesses average – it is ignored.

actor	day	#logins_during_msdb1	...
actor1	1	5	...
actor1	2	20	...
actor1	5	5	...
actor1	9	9	...
actor1	10	7	...
...	...	...	...

Aggregation by user account

actor	#distinct_days	sum_#logins_during_msdb1	mean_#logins_during_msdb1	std_#logins_during_msdb1	min_#logins_during_msdb1	q25_#logins_during_msdb1	median_#logins_during_msdb1	q75_#logins_during_msdb1	max_#logins_during_msdb1	...
actor1	5	46	9.2	6.26099	5	5	7	9	20	...
...	...	...	...	...	...	...	...	...	...	...

Figure 6.7: Illustrative example of the result obtained after the aggregation by user account. Due to confidentiality restrictions, the values of each feature were obfuscated.

6.3.2 Feature Engineering for Anomaly Detection

After user accounts are clustered based on their behavior, a list knowing to which peer groups a user account best belongs to is obtained. As previously mentioned, this project intends to detect deviations with respect to both corresponding account group (cross-sectional) and past (temporal) activity. To do this, it is required to create a single structured dataset containing information about the user account's daily activity, comparisons between the account's daily activity with the activity of all accounts within the same group (either on the same day as in the past), and comparison between the account's daily activity and its past activity. In addition, it is required that the insider-threat detection system be built based on the time unit day.

Using the daily dataset generated in Section 6.2, that is, a dataset that represents the account-day instance's activity by an array of frequency counts of meta-events, an extended version of the Robust Scaler has been applied to it. This method was chosen since it scales features using statistics that are robust to outliers. Although there is implementation of this method available in scikit-learn[1] that "removes the median and scales the data according to the quantile range" as required, it computes those relevant statistics considering all samples in the dataset. In the extended version developed, a range of subsets were also considered to meet the anomaly detection requirements. With more detail, in this extended version, the data was centered and scaled applying the RobustScaler to the following five (sub)sets:

- All the account's days in time period;

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

- All accounts in group i on same day;
- All accounts on all days in time period;
- All accounts on same day;
- All accounts in group i's days in time period;

As expected, the process described in this section produces two extremely high-dimensional datasets - one for Accounts Characterization and other for Anomaly Detection. However, many of their features correspond to either low-variance features or features that are correlated with each other. Applying the scikit-learn's feature selector *VarianceThreshold*[2] to each dataset, it was possible to remove features whose variance is (or is very close to) zero.
Thereafter, using the *.corr()* function[3], the correlation between features was calculated and plotted with the help of Plotly, that allows an interactive search. The Pearson correlation coefficient was used for the Accounts Characterization dataset. On the other hand, since correlation coefficient is not robust against outliers, the Spearman rank correlation coefficient was employed for the Anomaly Detection dataset. Once a high threshold is set, redundant features are removed.
Finally, the Accounts Characterization dataset was normalized, using the scikit-learn's Normalizer[4].

6.4 User Accounts Characterization and Anomalous User Accounts Detection

As mentioned before, one of the main tasks of this project is to discover a baseline population for comparison by grouping accounts based on their behavior. To accomplish this first goal, clustering algorithms were applied to the dataset describing accounts' profile that was built as described in the previous section. Those profiles are stored in a Dask DataFrame on which clustering algorithms, provided by the library *dask-ml*[5], will be employed. Since this library offers *k-Means* and *SpectralClustering* implementations, both will be implemented and, consequently, compared.
To find the set of parameters that best fit each model, an exhaustive grid-search hyperparameter tuning was performed. To be clear, since there is no classification of accounts available, nor certainty as to how they should be grouped, it is impossible to automate this process. With the analysis of the behavior of certain accounts, together with the help

[2]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html
[3]https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html
[4]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
[5]https://ml.dask.org/

of domain experts it was possible to know a restricted set of accounts that must be in the same group or in different groups, allowing the manual assessment of the accuracy of each model.
Additionally, to help assess the performance of each model, the following metrics that do not require knowledge about the ground truth have been applied: the Silhouette Coefficient[6], the Calinski-Harabasz index[7] - also known as the Variance Ratio Criterion -, and the Davies-Boulding index[8]. These metrics allow the assessment of how dense and well separated the clusters are. However, the evaluation of the quality of each cluster cannot be limited to the application of these metrics since they are not entirely suitable for density-based algorithms. Then, the algorithm and respective combination of parameters that best cluster those accounts is chosen.
This process not only allows accounts with similar behavior to be grouped, but also to detect accounts that have persistently abnormal behavior. This scenario may concern, for example, to accounts that are far from their cluster centroid, or close to the borders of the cluster, meaning that their behavior do not fit well in any cluster; small clusters; or clusters with unusual centroids. To do this, for each user account the distance to its cluster center, the silhouette value, as well as distance metrics (such as mean, standard deviation, minimum, first quartile, median, third quartile, and maximum) to user accounts of the cluster that belongs were calculated. After that, outliers against these metrics were detected using a box-plot for visualization and a rule that determines user accounts out of range of -1.5 x IQR to 1.5 x IQR. However, it is expected that most of the anomalies detected here are without malicious intent, but anomalies indicative of resource misuse or policy violation.

6.5 Anomaly Detection Module

As previously described, supervised, semi-supervised and unsupervised learning approaches are employed in anomaly detection tasks. However, since systems adopting supervised and semi-supervised methods require pre-labelled data which is a very costly and challenging task to achieve, we limited our analysis to unsupervised algorithms. Besides that, as no single anomaly detection method stands for universal applicability, choosing a suitable anomaly detection technique among the several available options meeting the specific application requirements is a challenging and essential task. The selection of an inappropriate algorithm leads to decreased attack detection efficiency, exhibiting larger rates of missed (False Negatives) and wrong detections (False Positives). Therefore, it is essential to the completeness of this work to choose a set of algorithms covering all the core categories of the unsupervised anomalous detection techniques: clustering-based,

[6]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
[7]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabasz_score.html
[8]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html

density-based, neighbor-based and model-based methods.
In this dissertation, multiple unsupervised anomaly detectors were employed and compared. Each of these detectors is subject to a choice of model, choice of input feature and hyper-parameter settings. Although the ensemble methods have been proven to be effective and robust in classification problems, there is a scarcity of well-researched and formalized ensembles for unsupervised anomaly detection. The main goal of ensemble learning is to combine the output of multiple base learners in order to outperform all of single learners. However, as already mentioned, one of the great challenges is the lack of ground truth and, consequently, of evaluation methods for anomaly detection methods in general. As such, being an active and challenging research topic in the area of anomaly detection for which there are no strong conclusions yet, an approach based on ensemble learning will not be adopted in this project.
Once the feature set for the user accounts' current daily observations has been computed, the following five unsupervised anomalous detection techniques were performed (using the implementation provided by the Python library *PyOD*[9]) and compared: k-Nearest Neighbor (kNN), One-Class Support Vector Machines (One-Class SVM), Local Outlier Factor (LOF), Cluster-Based Local Outlier Factor (CBLOF), and Isolation Forest (iForest).
As it happened when Clustering methods were applied, an exhaustive grid-search hyper-parameter tuning was performed to find the set of parameters that best fit each model. In addition to trying to find the optimal combination of parameters for each algorithm, an exhaustive search was also made to select a set of effective feature weighting that guarantee better performance. This is a crucial task because not all information is equally important for anomaly detection. Recalling the example presented in *R4.2)* of Chapter 4 where an account that, despite having an anomalous behavior with respect to its past activity, has a normal behavior with respect to the accounts of the same group and therefore is not detected as an anomaly, it is clear that under these circumstances the information on the behavior of the accounts of the same group is supposed to overlap the account's past behavior information. Additionally, there are features with greater importance to the problem (for instance, related to an anomaly metric) that should be emphasized.
This exhaustive search was not done considering all possible weight combinations, but combinations that give more importance to features that contribute the most to the main components when applying the Principal Component Analysis (PCA) algorithm. As previously described, each feature can have information about the group, the user or all users (first set) as well as with the action, location, range_type, executable, number or hour (second set). Figure 6.8 outlines this relationship.
The strategy adopted to assign weights to features was firstly to give weights to the disjoint subsets of the first and second sets such that their sum was equal to 1. After that, the

[9] https://pyod.readthedocs.io/en/latest/

weight of a feature *f*, $w(f)$, is given by the following expression:

$$w(f) = w_{set1}(f)/(6 * len(set1(f), set2(f))) + w_{set2}(f)/len(set1(f), set2(f)))$$

where $w_{set1}(f)$ and $w_{set2}(f)$ are the weights associated with the subset of set 1 and 2 to which the feature f belongs, respectively, and $len(set1(f), set2(f)))$ is the number of features belonging to the same set 1 and 2 as f.

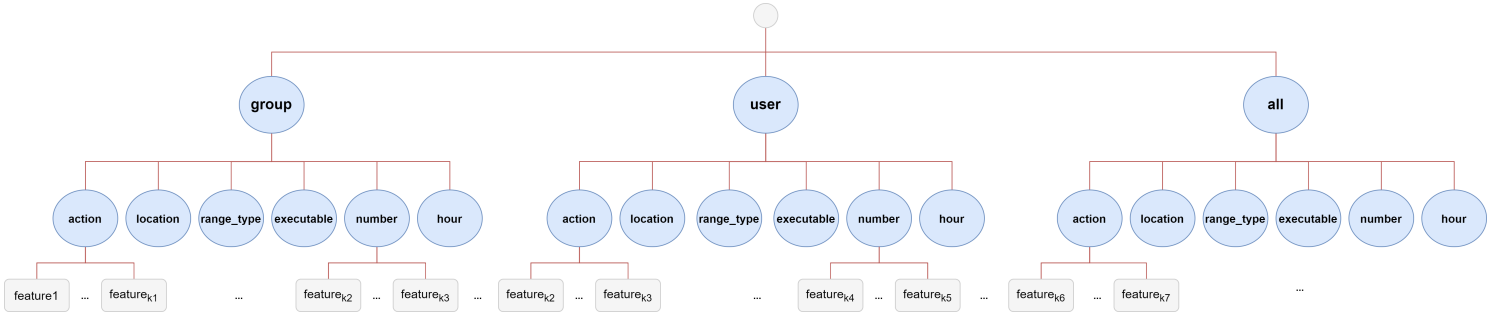


Figure 6.8: Illustrative example of the relationship between sets of information given by the features.

Due the large size of the dataset under study, it is expected that most of the conventional algorithms, such as nearest neighbour- and clustering-based techniques, have a lower detection rate or an abrupt increase in the time complexity. On the other hand, algorithms belonging to the model-based family, such as Isolation Forest and One-Class SVM, are expected perform more efficiently and produce better scores. In fact, nearest-neighbor methods are expected to perform better for a global task, and for a local task, is likely that the Local Outlier Factor algorithm to be more suitable than clustering-based techniques.
Each of these detectors assigns a normalized anomaly assessment score to each of the account-day pair. Defining a threshold on the decision function, an anomaly score below this threshold is assumed normal, otherwise, is flagged as anomalous. However, since the amount of contamination of the dataset, in other words, the proportion of anomalies in the dataset, is not known, choosing this threshold is not an easy task.
In order to evaluate each anomaly detector performance and to choose a good anomaly threshold, the following ten simulated insider threat scenarios were performed and injected in the dataset:

- Anomaly 1 - Nominal account, which typically only access during working hours, with accesses after working hours in a day.
- Anomaly 2 - Nominal account with some accesses at the weekend.
- Anomaly 3 - Nominal account that only makes *logins* to perform *reads* in a day.

- Anomaly 4 - Nominal account, which typically only has *rtype1* and *rtype2* as actor_range_type of the accesses, having actor_range_type *rtype3* in a day. The three distinct values of this feature are represented by *rtype1*, *rtype2* and *rtype3* for the sake of confidentiality.

- Anomaly 5 - Nominal account with accesses from the three actor_range_type values (*rtype1*, *rtype2* and *rtype3*) on the same day.

- Anomaly 6 - Nominal account actor_executable *exec1* instead of *exec2*, which is the expected for this type of account, in a day. For the same reason mentioned before, the two distinct values of actor_executable feature will be represented by *exec1* and *exec2*.

- Anomaly 7 - Nominal account with a very large number of accesses (almost impossible to be reached by humans) on the same day.

- Anomaly 8 - Nominal account with multiple location accesses on the same day. As a rule, a nominal account only accesses from one location.

- Anomaly 9 - Application account, which typically only has *rtype3* as actor_range_type of the accesses, having actor_range_type *rtype1* in a day.

- Anomaly 10 - Application account with accesses from the three actor_range_type values (*rtype1*, *rtype2* and *rtype3*) on the same day.

Note that the scenarios created are some that security experts think that are likely to happen, however this set does not represent the complete set of real insider threat scenarios. It is required to detect scenarios that have not previously been seen or unknown by the developers of the system. To do this, for each insider threat scenario a user account without abnormal behavior that meets each anomaly's requirements (in Anomaly 1, for instance, a nominal account, which typically only access during working hours) was identified. This selection was done through the combination visualization techniques with the help of security experts. For each chosen account, a duplicate account is generated and the behavior of one of its active days is modified in order to produce the insider threat scenario that it is intended to represent.
Thus, both the injected insider threat scenarios and the identified accounts without abnormal behavior were used to evaluate the anomaly detectors performance, as well as to choose a good anomaly threshold.
Given the volume of the dataset, choosing the threshold with minimal false positives is a complex and time consuming task and would also require the participation of Security Operations Center (SOC) analysts. Therefore, despite not guaranteeing optimality, the approach adopted - which has proved to be successful - is to run several detectors with a

high number of distinct contamination proportions between 0 and 0.5 and to select the detector with minimal contamination that best classifies both the injected anomalies and the identified normal instances, as well as instances near the boundary of this threshold. This assessment is made in ascending order of contamination proportion, since it is expected that a value close to 0.5 will generate a very high number of false positives.

6.6 Anomaly Characterization Module

Once an anomaly is detected, it is intended to be able to characterize its anomalous scenario to help identify the risk it poses to the company. Since there is no knowledge on the part of Altice's security experts about a specific set of anomalies that are intended to direct this investigation, the purpose of this component is to characterize each anomaly as to its nature, identifying what particular attributes caused the anomaly to be detected. This allows analysts to identify those who represent malicious attacks and to identify those which they should focus on. This can be achieved by creating a set of anomaly metrics combined with visual analytics tools provided by Kibana.

An anomaly metric indicates whether an instance - a (user, day) pair - is anomalous (1) or not (0) for that metric. The following nine anomaly metrics have been implemented: Action_anomaly, Location_anomaly, Range_type_anomaly, Executable_anomaly, Number_anomaly, Hourly_anomaly, Account_anomaly, Group_anomaly, and All_anomaly. For instance, Action_anomaly indicates whether an instance is anomalous (1) or not (0) by the type of action(s) performed. An anomaly detected in this project could be associated with either performing an activity that is anomalous compared to past account activity, or performing an activity that is anomalous compared to the activity of the accounts of the same group, or performing an activity that is anomalous compared to the activity of all accounts. These define the "Account," "Group," and "All" anomaly metrics, respectively. Each of these anomaly metrics is given by applying the process performed in the previous section to the subset of the overall feature set that are related to the context of the metric. For example, if a user account performs an existing activity at a new hour of the day, the Hourly_anomaly metric value is expected to be high. To do this, the set of features that give information about an activity has been done during or after work hours is selected and the grid-search for the detector that produces better performance is done as before. Furthermore, the same instances used in Section 6.5 were also adopted here to evaluate the detectors. Table 6.1 shows which anomaly metrics must be triggered for each of the ten injected anomalies presented before.

The final stage is to build a matrix combining the collection of anomaly metrics that have been assigned to each detected daily account activity profile anomaly. Having said that, for each daily account activity profile detected as anomalies (rows) and for each anomaly metric (columns), the corresponding binary label - 0 for inliers and 1 for anomalies – is

associated. As already mentioned, is essential to look at the value of the anomaly metrics to understand nature of an anomaly. For instance, if an account-day instance has a value of 1 in Hourly_anomaly, Location_anomaly, Account_anomaly and Group_anomaly, and 0 in the others, it means that the current account on the current day was accessed from an abnormal location and time, given its past activity and the activity of the group of accounts it belongs to.

Table 6.1: Anomaly metrics associated with each of the injected anomalies.

Anomaly	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Action_ anomaly			X							
Location_ anomaly								X		
Range_type_ anomaly				X	X				X	X
Executable_ anomaly						X				
Number_ anomaly							X			
Hourly_ anomaly	X	X								
Account_ anomaly	X	X	X	X	X	X	X	X	X	X
Group_ anomaly	X	X	X	X	X	X	X	X		X
All_anomaly					X					X

In addition, combining multiple visualization techniques, a dashboard to compare the behavior of the detected account with the behavior of other accounts and another to compare with its past behavior were built in order to enable a quick and easy identification of the detected anomaly, as well as the risk it poses to the company. The first dashboard contains line charts with the frequency of values (count) over time and frequency tables comparing the values of the *action*, *action_details*, *actor_range_type*, *actor_executable*,*actor_location* and *db_brand* features for the selected user accounts. The second dashboard contains a line chart with the frequency of accesses (count) over time, pie charts, stacked bar charts with the frequency of values (%count) over time, and frequency tables comparing the values of the *action*, *action_details*, *actor_range_type*, *actor_executable*,*actor_location* and *db_brand* features for the user account under analysis. These dashboards can be seen in Figures 6.9 and 6.10, respectively. Thus, it will be possible to take measures to prevent possible damage to the company.

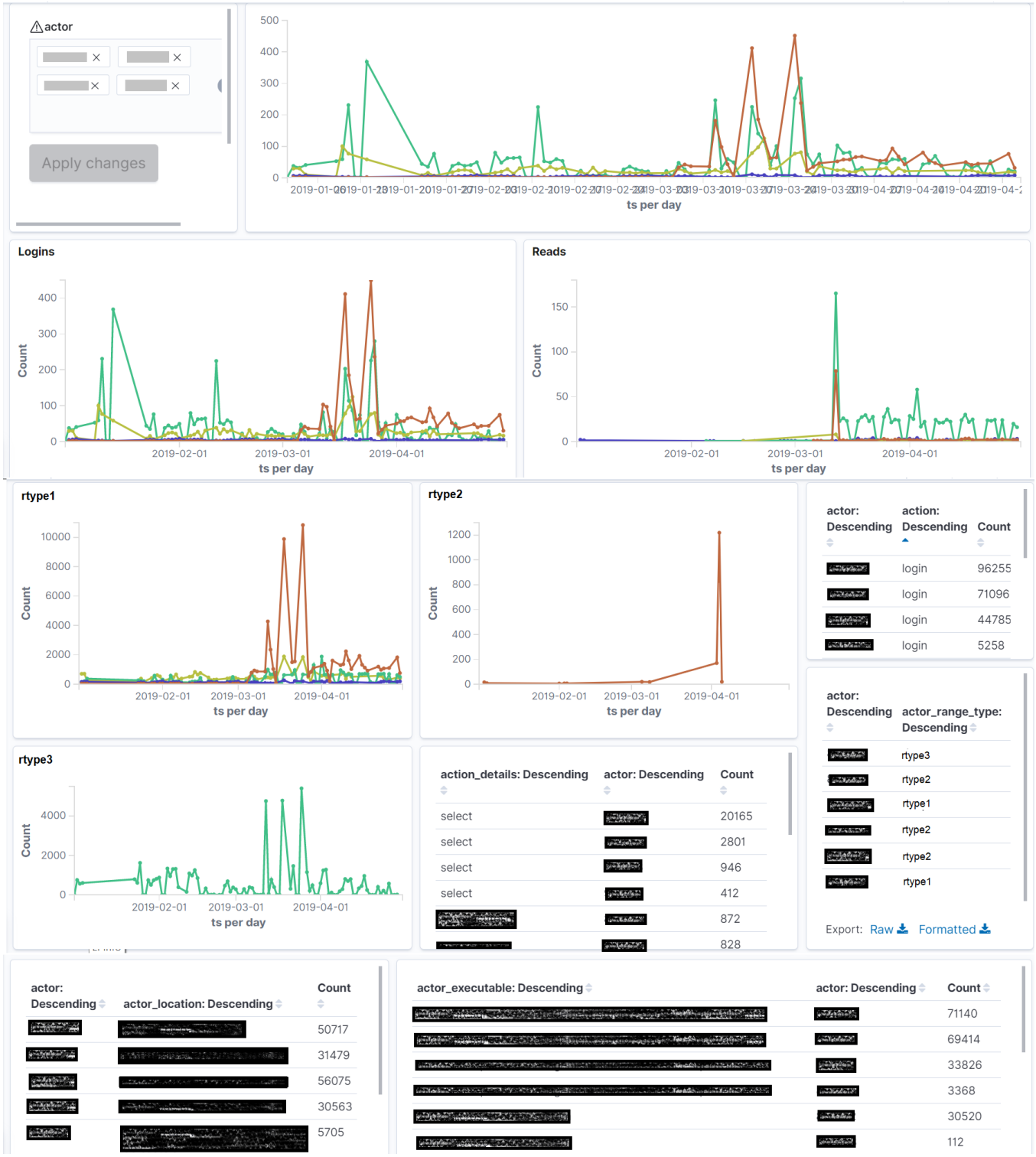


Figure 6.9: Dashboard for comparison of user accounts behavior across multiple attributes and domains.

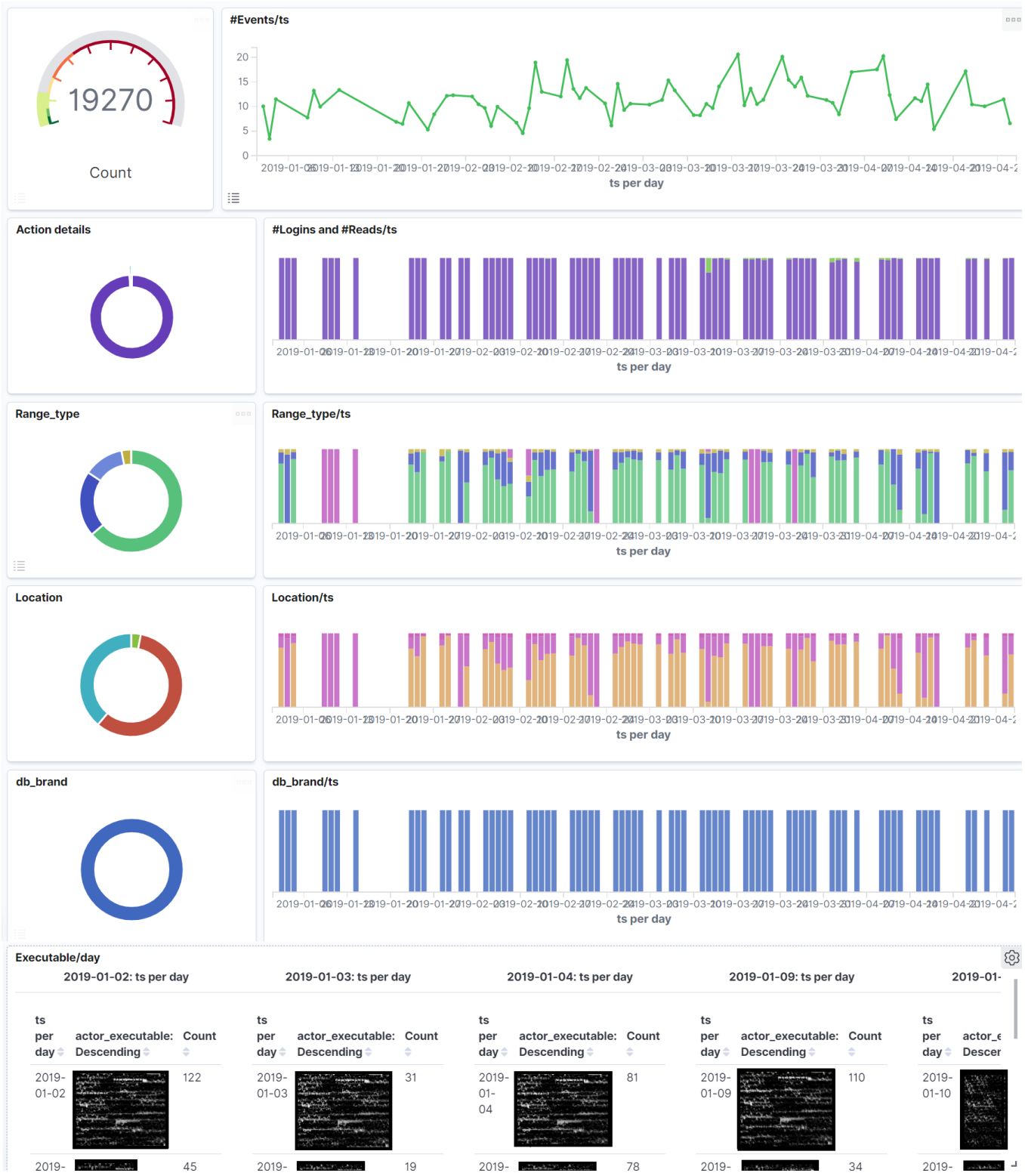


Figure 6.10: Dashboard for temporal comparison of individual user account activity on multiple attributes and domains.

Chapter 7

Results and Discussion

In this chapter, experimental results for each system modules previously described will be presented as well as a brief description of the type of anomalies detected. Note that, for the sake of space limitations, only the most relevant and representative examples will be displayed. In addition, due to confidentiality, some values will be omitted or obfuscated.

7.1 Exploratory Data Analysis and Feature Selection

The first step was to understand the business and the problem being tracked, as well as the information available for its resolution, through exploration and analyzing data with the help and supervision of domain experts. As such, some results from this process will be presented below.

1. Variable Datatype Identification - First, we performed a quick inspection of feature values from a small subset of records, as well as the identification of the datatype of each of them, as shown in Table 7.1.

Table 7.1: Variable Datatype Identification

Variable Name	**Datatype**
action	Categorical
action_details	String
actor	String
actor_account_risk	Categorical
actor_account_state	Categorical
actor_company	String
actor_department	String
actor_details	String
actor_domain	String
actor_executable	String

Continued on the next page

Table 7.1: Variable Datatype Identification (cont.).

Variable Name	Datatype
actor_identity_id	Categorical
actor_identity_risk	Categorical
actor_ip	String
actor_ip_as_name	String
actor_ip_hostname	String
actor_ip_range_type	Categorical
actor_location	Categorical
actor_location_id	Categorical
actor_network_protocol	Categorical
actor_network_range	String
actor_network_type	Categorical
actor_range_type	Categorical
actor_type	Categorical
db_brand	Categorical
object	String
object_group	Categorical
object_ip	String
ts	Datetime

2. Univariate Analysis - Since we are dealing with categorical features, in order to understand their distributions and explore them, frequency tables based on two metrics, *count* and *%count*, and bar charts were created. Tables 7.2 and 7.3 present the frequency tables of the features actor_range_type and db_brand, respectively.

Table 7.2: actor_range_type frequency table

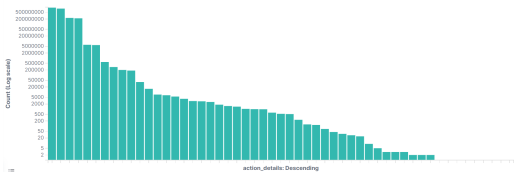
Distinct values	Count	%Count
dtype1	3 687 842	0.22
dtype2	701 581	0.04
dtype3	1 666 186 105	99.74

Table 7.3: db_brand frequency table

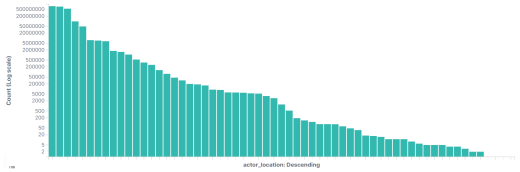
Distinct values	Count	%Count
db_brand1	976 567 370	44.08
db_brand2	623 814 517	28.16
db_brand3	597 173 568	26.95
db_brand4	17 996 104	0.81

Figure 7.1 presents an analysis of the frequency distribution in logarithmic scale of values for some features in the dataset. As we can see in Figure 7.1a, there is a great disparity in terms of access volume for each account. This volume is expected to be dependent on the type of account, so that accounts with a very large volume correspond to application accounts, while those with a small volume correspond to human user accounts. In order to understand if the feature value frequency is related to multiple accounts, or just a few accounts with a large volume of accesses, an analysis of these values according to each account was made as shown in the Figure 7.2.

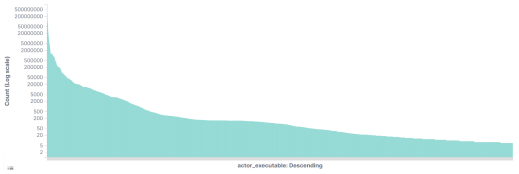
(a) Number of access log records per user account (*actor* value).



(b) Number of access log records per action performed (*action_details* value).

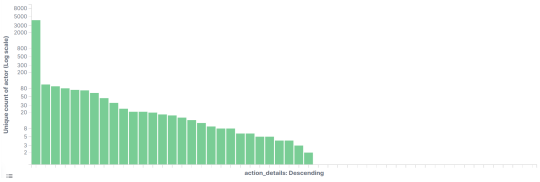


(c) Number of access log records per access location (*actor_location* value).

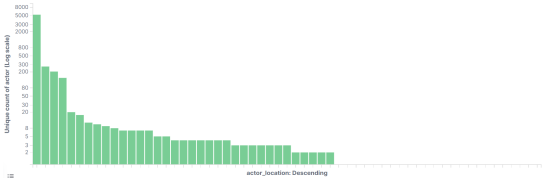


(d) Number of access log records per *actor_executable* value.

Figure 7.1: Absolute frequency distribution in logarithmic scale of features' values.



(a) Number of user accounts per *action_details* value.



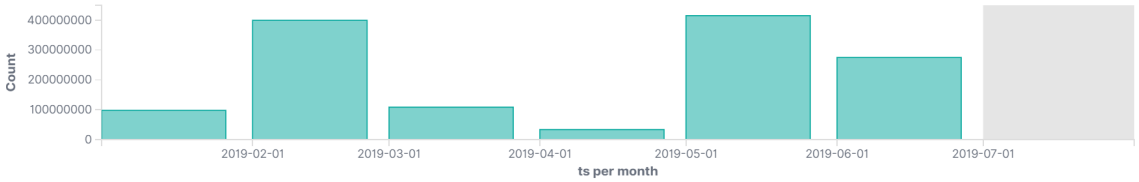
(b) Number of user accounts per *actor_location* value.

(c) Number of user accounts per *actor_executable* value.

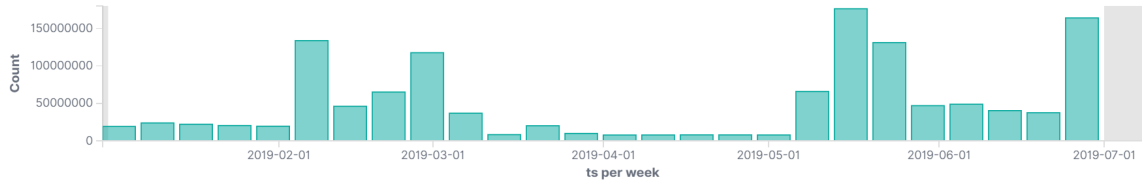
Figure 7.2: Absolute frequency distribution in logarithmic scale of features' values according to user accounts (*actor*).

Figure 7.3 shows the distribution of feature *ts* values of the full dataset considering different time granularity (month, week, day, and hour, respectively). As expected, when lower

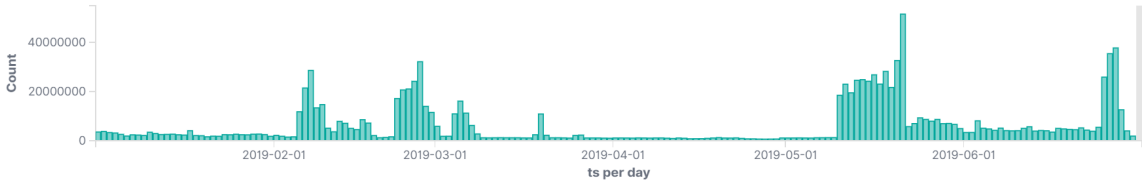
levels of detail (more granular data) are being exploited, better analytical capabilities are achieved. However, it is also crucial to consider the increase in computational complexity and time. As mentioned earlier, day was the time granularity chosen.



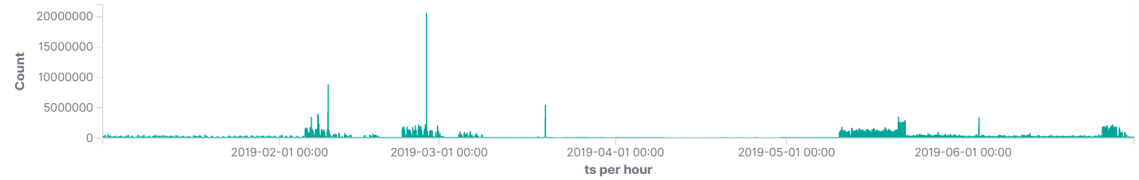
(a) Number of access log records per month.



(b) Number of access log records per week.



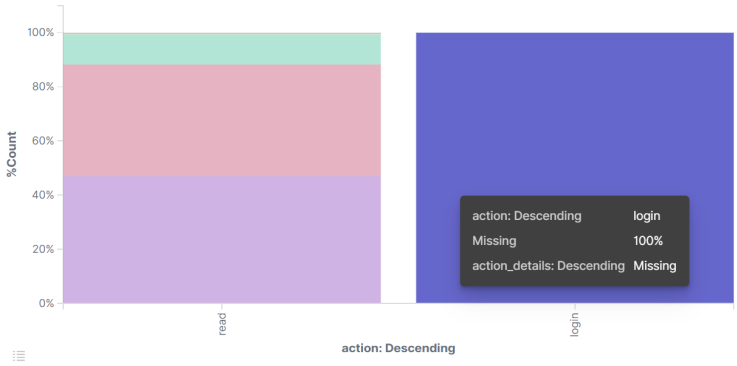
(c) Number of access log records per day.



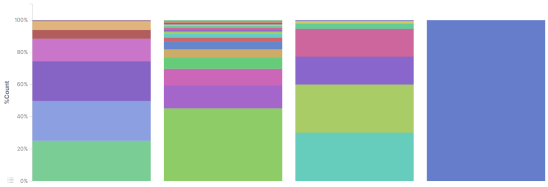
(d) Number of access log records per hour.

Figure 7.3: Absolute frequency distribution of *ts* feature values considering distinct time granularity.

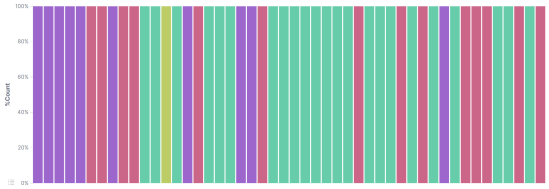
3. Bi-variate Analysis - After analysing each feature individually, relationships between two features at a predefined significance level were analyzed. To do that, stacked column charts of count, %count, unique count of actor and %unique count of actor were created. Figure 7.4 shows the most relevant stacked column charts of %count built comparing the values of features pairs (one by columns and other by colors). For confidentiality reasons these values were hidden, however some correlations were identified. As we can see in Figure 7.4a, *login* values in *action* correspond to missing values in *action_details* (and vice versa) and all non-null values correspond to *read* actions. Additionally, Figures 7.4b and 7.4c show a relationship between two object granularity levels as illustrated in the Figure 6.3 emphasizing that an application (*object_group*) is associated to only one database



(a) Relationship between *action* and *action_details*



(b) Relationship between *db_brand* and *object_group* (*object_group* as a function of *db_brand*)



(c) Relationship between *db_brand* and *object_group* (*db_brand* as a function of *object_group*)

Figure 7.4: Stacked column plots analyzing relationships between values of two features.

management system (*db_brand*), but a database management system can be associated to one or more applications.

4. Missing and Noisy Values Identification - The final task is to identify the missing values and outliers, as well as the reasons for their occurrences, whether its cause is natural or artificial essentially, in order to find a better way to deal with them.

Table 7.4: Missing and Noisy Data Identification

Variable Name	#Missing data	%Missing data	#Noisy values	#Noisy instances
action	0	0.00	0	0
action_details	231 193 980	10.45	1	163 245
actor	13 042 869	0.59	1 252	48 034 090
actor_account_risk	1 734 370 323	78.28	0	0
actor_account_state	1 734 370 323	78.28	1	17
actor_company	2 101 272 024	94.84	0	0
actor_department	2 106 500 360	95.10	0	0
actor_details	1 455 130 433	65.68	15	21
actor_domain	1 918 389 684	86.59	2	4
actor_executable	527 738 964	23.82	0	0
actor_identity_id	942 917 778	42.56	6	991 165 660

Continued on the next page

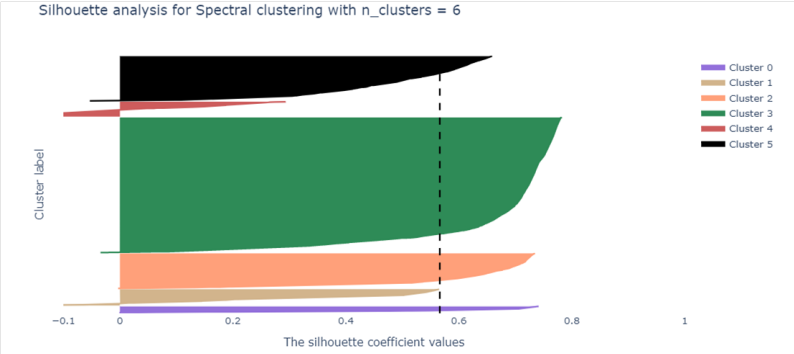
Table 7.4: Missing and Noisy Data Identification (cont.).

Variable Name	#Missing data	%Missing data	#Noisy values	#Noisy instances
actor_identity_risk	1 682 081 627	75.92	0	0
actor_ip	115 477 145	5.21	1	17 672 990
actor_ip_as_name	2 215 551 397	99.99	0	0
actor_ip_hostname	844 939 428	38.14	0	0
actor_ip_range_type	497 324 019	22.45	0	0
actor_location	518 468 047	23.40	0	0
actor_location_id	533 807 242	24.09	0	0
actor_network_protocol	0	0.00	0	0
actor_network_range	480 333 060	21.68	0	0
actor_network_type	875 190 402	39.50	0	0
actor_range_type	539 259 776	24.34	0	0
actor_type	941 785 232	42.50	0	0
db_brand	0	0.00	0	0
object	0	0.00	0	0
object_group	776 091 243	35.03	1	2 395
object_ip	0	0.00	0	0
ts	0	0.00	0	0

Once the exploratory data analysis was carried out and presented to the domain experts, the relevant attributes for the problem to be solved were chosen. Therefore, a first experiment considering these features and all dataset records was implemented. Figure 7.5 shows the results obtained after the clustering module of the user accounts. After that, the anomaly detection component was performed, however, given the size of the dataset, detectors' training, evaluation and comparison were time consuming tasks.

As mentioned earlier, there are user accounts for different functions and not all anomalies are interesting for this business context. In fact, anomalies with a higher risk are those that are directly associated with human behavior. As such, service and network accounts are not as relevant as personal user accounts.

Based on the results obtained from the clustering, it was possible with the help of experts to find filters that would reduce the size of the dataset and that would direct our investigation. A set of accounts with a potential high insider threat risk are those belonging to cluster 0 (*purple*). Analyzing the type of database they accessed, the target data chosen was the accesses to objects belonging to a given application (*object_group*). From now on, the results presented in the following sections refer to the dataset with these selected data and features. Note that the process applied considering all records is analogous.



(a) Silhouette plot for the selected Clustering model

(b) Visualization of the six clusters obtained through a dimensionality reduction into a two-dimensional space using t-SNE.

Figure 7.5: Visualization of the results obtained by the Characterization of User Accounts Module.

7.2 Data Integration, Preprocessing and Preparation Pipeline

Table 7.5 presents the number of distinct values in each extracted feature when considering the entire dataset before and after Data Cleaning. Additionally, Table 7.6 shows the size reduction and/or dimensionality increase after each task involved in this module have been performed.

Table 7.5: Number of distinct values for each feature before and after Data Cleaning.

Feature	**Before**	**After**
actor	7 133	5 881
ts	46 813 554	181
db_brand	4	4
action_details	52	10
actor_location	60	47
actor_executable	10 113	3
actor_range_type	3	4

Table 7.6: Dataset size and dimensionality after running each step involved in the Data Extraction, Preprocessing and Preparation module. The number in * is not shown since it was an intermediate step and the generated dataset was not saved, due to space limitations.

Step	#Instances	#Features
Initial dataset	81 720 409	7
After Data Cleaning	81 720 188	19
After Feature Extraction	*	19
After Unstack during_work	59 733	32
After Unstack db_brand (Final)	15 298	118

7.3 Data Transformation

7.3.1 Feature Engineering for User Accounts Characterization

To discover groups of user accounts with similar behavior, it was required to create a dataset summarizing the behavior of each account during the six months. Figure 7.6 exemplifies user accounts with the same number of records during the six months, however with different daily behavior. Therefore, the dataset was grouped by user account not only computing metrics summing features' values for those groups, but also daily metrics such as mean, standard deviation, minimum, first quartil, median, third quartil, and maximum. Table 7.7 shows the size and dimensionality variation after each step involved in building the dataset for clustering have been performed. As expected, there is a great dimensionality reduction after applying the Variance Threshold method, since this subset concerns access to a specific *object_group* that belongs to a (single) *db_brand*. Due to that, approximately three quarters of the features have zero variance.

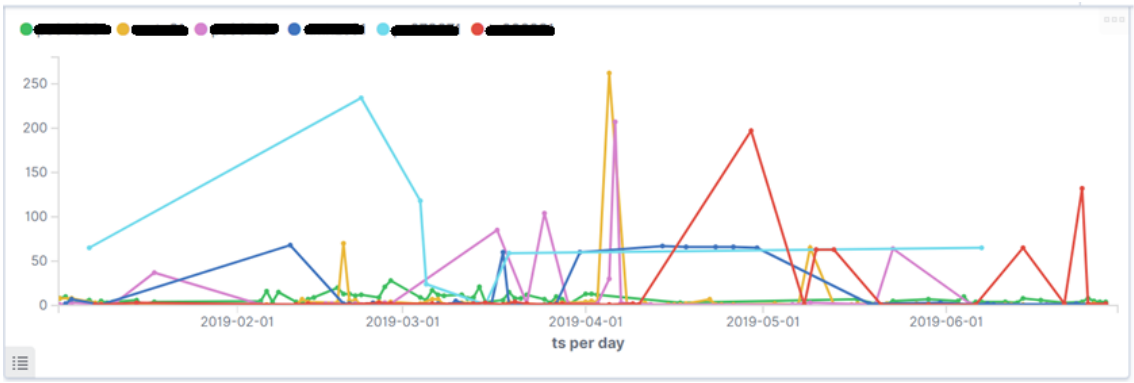


Figure 7.6: Daily count of access log records of six user accounts with the same total number of records.

Table 7.7: Dataset size and dimensionality after running each step involved in the Feature Engineering for Clustering component.

Step	#Instances	#Features
Initial dataset	15 298	118
After Feature Engineering	204	930
After Heuristic of Distinct Days	130	926
After Variance Threshold	130	192
After Removing Correlations	130	23
After Normalization (Final)	130	23

7.3.2 Feature Engineering for Anomaly Detection

Once obtained the list with the group to which each user account belongs to, this information is combined with the dataset constructed in Section 7.2. Table 7.8 presents the transformations made in dataset instances or features after each step involved in the dataset construction for anomaly detection process.

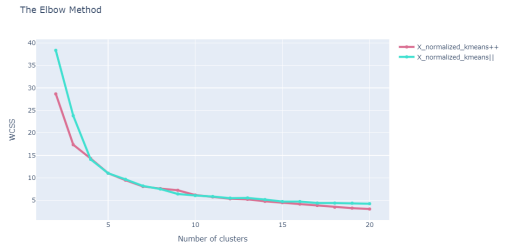
Table 7.8: Dataset size and dimensionality after running each step involved in the Feature Engineering for Anomaly Detection component. Note that dimensionality reduction through *VarianceThreshold* and *Spearman Correlation* were additionally performed in intermediate steps (*) to avoid generating a large number of unnecessary (intermediate) features.

Step	#Instances	#Features
Initial dataset (actor, day)	15 298	118
After Merging Clustering Results	14 919	119
After Injecting Anomalies	15 174	119
After Variance Threshold*	15 174	32
After Ratio Features Creation	15 174	34
After Removing Correlations*	15 174	20
After Extended RobustScaler	15 174	88
After Variance Threshold	15 174	75
After Removing Correlations (Final)	15 174	54

7.4 User Accounts Characterization and Anomalous User Accounts Detection

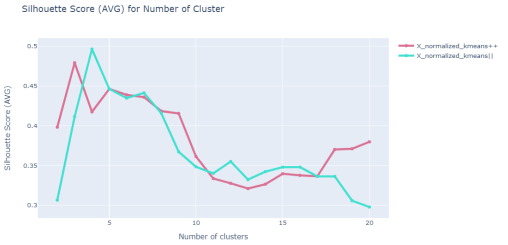
In this section, in order to group and characterize user accounts with similar behavior, both the k-Means and Spectral Clustering approaches were considered and subsequently compared. The full list of parameterizations explored can be found in Appendix A.
The evaluation metrics values obtained for each run of the k-Means and Spectral Cluster-

ing algorithms for each combination of parameters are presented in Figures 7.7 and 7.8, respectively. Note that plots were filtered considering the initial number of clusters. As mentioned earlier, these metrics are not suitable when considering density-based algorithms, so a manually and qualitative evaluation strategy supported by security analysts with knowledge of the behavior of each account was adopted. As expected, the Spectral Clustering showed to obtain better results.

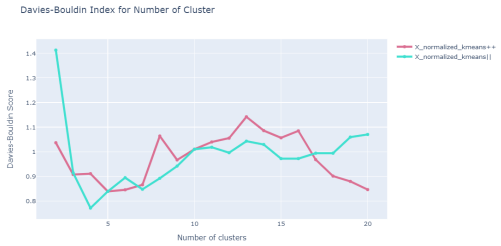


(a) The Elbow Method.

(b) Calinski-Harabasz Index.

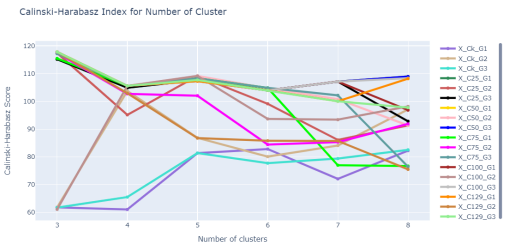


(c) Silhouette Coefficient.



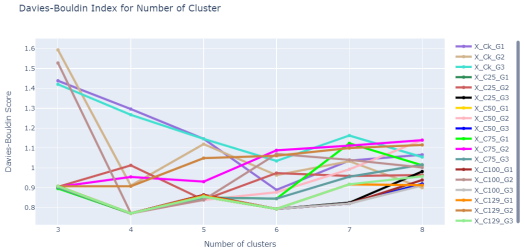
(d) Davies-Bouldin Index.

Figure 7.7: Performance evaluation metrics for k-Means algorithms.



(a) Calinski-Harabasz Index.

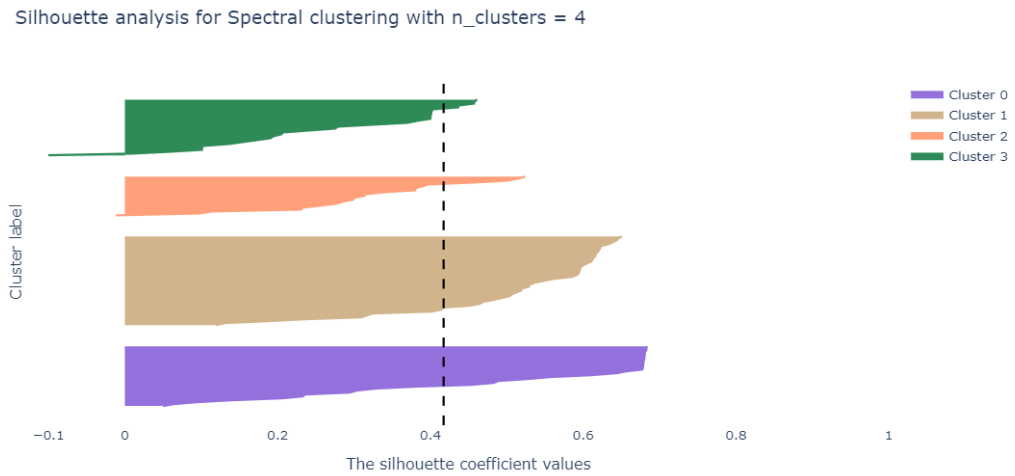
(b) Silhouette Coefficient.



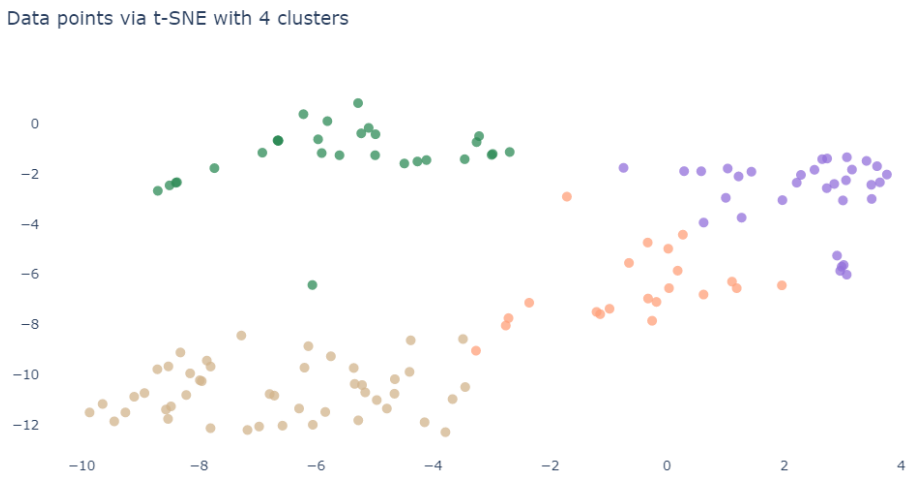
(c) Davies-Bouldin Index.

Figure 7.8: Performance evaluation metrics for Spectral Clustering algorithms.

Figure 7.9 presents a detailed analysis of the clustering of similar user accounts obtained. Analysing the features values from user accounts belonging to each cluster, it was possible to differentiate the accounts of each cluster as to their function. Both clusters 0 and 3 can be identified as application accounts, and both clusters 1 and 2 as nominal accounts. However, clusters of nominal and application accounts can be differentiated both in terms of access volume, as well as in terms of actions taken. For confidentiality reasons, the description of each cluster will not be presented.



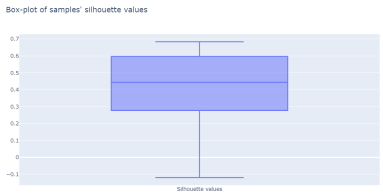
(a) Silhouette plot for each cluster.



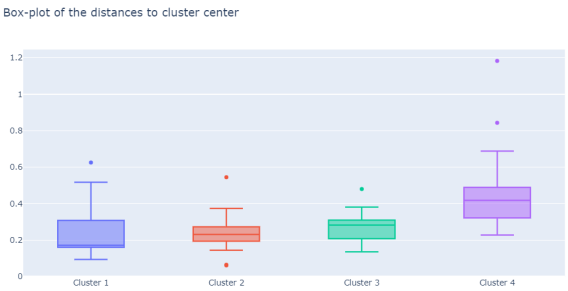
(b) Visualization of the clustered data via t-SNE.

Figure 7.9: Analysis of the clustering of similar user accounts obtained.

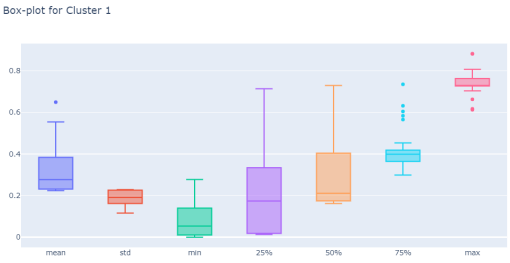
Once chosen the algorithm and respective combination of parameters that best cluster user accounts, a list of user accounts that have persistently abnormal behavior was achieved. The box-plots obtained during this analysis are displayed in Figure 7.10.



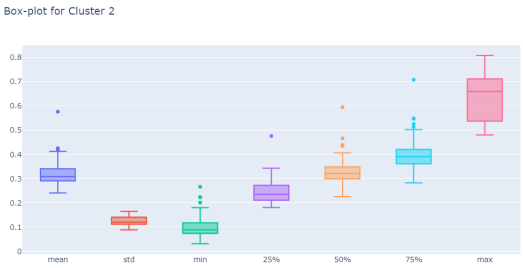
(a) Box-plot of user accounts silhouette values.



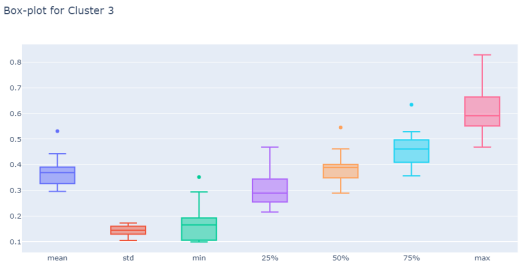
(b) Box-plots for each cluster of user account distances to its center.



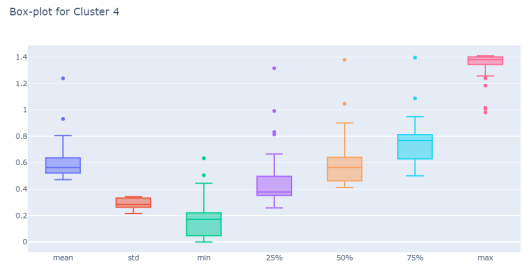
(c) Metric box-plots of distances between user accounts belonging to Cluster 1.



(d) Metric box-plots of distances between user accounts belonging to Cluster 2.



(e) Metric box-plots of distances between user accounts belonging to Cluster 3.



(f) Metric box-plots of distances between user accounts belonging to Cluster 4.

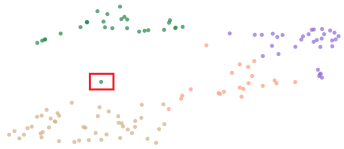
Figure 7.10: Box-plots to aid detection of user accounts that have persistently abnormal behavior.

Figure 7.11 shows the behavior of some user accounts detected. As expected, most of the anomalies detected are without malicious intent, but anomalies indicative of resource misuse or policy violation, for example, the use of the same account for application and nominal purposes.

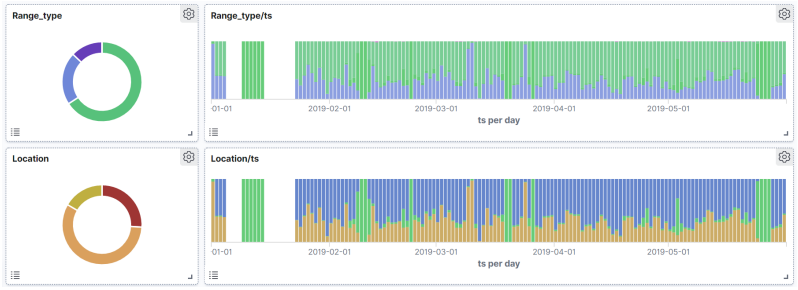
Although these accounts have a behavior that differs from the others, this behavior is normal with respect to the past behavior of each account. Thus, they are maintained for the following stages as they may have an anomaly with a malicious character that deviates from their behavior and that has not been identified yet.

(a) Example of a detected user account belonging to Cluster 1.

(b) Example of a detected user account belonging to Cluster 2.



(c) Visualization of a detected user account beloging to Cluster 3 by t-SNE.



(d) Example of a detected user account belonging to Cluster 3.

(e) Example of a detected user account belonging to Cluster 4.

Figure 7.11: Examples of detected user accounts that have persistently abnormal behavior.

7.5 Anomaly Detection Module

This exhaustive grid-search hyper-parameter tuning is performed after the simulated insider threat scenarios are injected into the dataset resulting from Section 7.3.2. The full list of parameterizations and feature weights explored can be found in Appendix B. As mentioned earlier, to simplify this task, the evaluation of each detector was made by comparing the returned binary vector, instead of considering the outlier scores of the training data. Table 7.9 shows the injected anomalies detected by the best detectors based on each of the Anomaly Detection algorithms implemented. Note that during the evaluation process, real anomalies were identified and also used in this process. As expected, algorithms belonging to the model-based family, such as Isolation Forest and One-Class Support Vector Machines, performed more efficiently and produced better results. In fact, it was the One-Class Support Vector Machines algorithm that produced the best result.

Table 7.9: Anomalies detected by each Anomaly Detection algorithm.

Anomaly	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
kNN	X	X	X	X		X		X		
OCSVM	X	X	X	X	X	X	X	X	X	X
LOF	X	X	X	X		X		X	X	
CBLOF	X	X	X			X			X	
iForest	X	X	X	X	X	X		X	X	

7.6 Anomaly Characterization Module

As mentioned earlier, the investigation of the detected anomalies is aided through the creation of some anomaly metrics that are determined by the execution of the anomaly detection process previously presented considering a subset of features related to the respective anomaly metric. Table 7.10 shows the dimension of the feature subset of each anomaly metric.

Table 7.10: Feature subset dimension of each anomaly metric.

Anomaly Metric	**#Features**
Action_anomaly	12
Location_anomaly	4
Range_type_anomaly	26
Executable_anomaly	14
Number_anomaly	3
Hourly_anomaly	17

Continued on the next page

Table 7.10: Feature subset dimension of each anomaly metric (cont.).

Anomaly Metric	#Features
Account_anomaly	16
Group_anomaly	20
All_anomaly	15

As previously verified, both Isolation Forest and One-Class Support Vector Machines algorithms performed more efficiently and produced better results for these subsets.

7.7 Detected Anomalies Examples

Finally, in this section a significant set of examples of distinct types of anomalies detected by the model is presented and discussed, as well as dashboard visualizations that allow human analysts to easily detect them.

The first anomalous scenario refers to volumetric changes in user account accesses, given by the anomaly metric *Number_anomaly*. Figures 7.12 and 7.13 show an example of a nominal and application account whose access volume is extremely high on that day (red) compared to its past behavior, respectively.

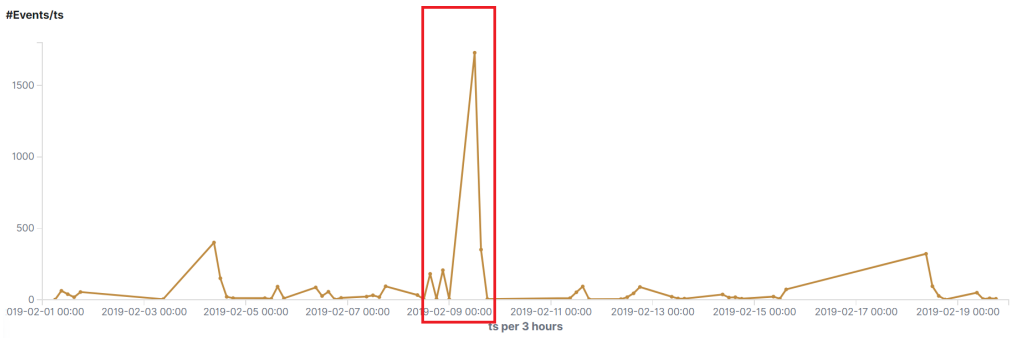


Figure 7.12: Example of detected anomalies 1 - Nominal user account with abnormal access volume (*Number_anomaly*).

Figure 7.13: Example of detected anomalies 2 - Application user account with abnormal access volume (*Number_anomaly*).

The second anomalous scenario concerns with user account behavior deviations from their own previous behavior, represented by the anomaly metric *User_anomaly*. Figure 7.14 shows an example of an application user account with two different types of daily behavior depending on whether it is a day of the week (blue) or not (light red), having been detected an anomalous day (red) deviating from expected (similar to the days in *blue*) behavior (*User_anomaly*).

Figure 7.14: Example of detected anomalies 3 - Application user account with abnormal behavior comparing to its past activity (*User_anomaly*).

Like the previous one, this anomalous scenario refers to user account behavior deviations from their own previous behavior (*User_anomaly*), however in this case it is caused by a deviation in the access executable, given by the anomaly metric *Executable_anomaly*. Figure 7.11 shows an example of a nominal user account whose executable value is always *user* (*User_anomaly*) but that has a day with an (anomalous) *application* access (bold row).

Another anomalous scenario concerns with the time the accesses were made, considering whether they happen after normal working hours or not, that is identified by anomaly metric *Hourly_anomaly*. Figure 7.12 presents an example of a nominal user account who usually only access during work hours (from 8 a.m. to 6 p.m.) and accesses on after work hours in two days (bold rows).

Date	#User	#App	#Missing
May 2, 2019	493	0	0
May 3, 2019	65	0	0
May 6, 2019	230	0	0
May 7, 2019	311	0	0
May 9, 2019	259	0	0
May 10, 2019	106	0	0
May 13, 2019	427	0	0
May 14, 2019	945	0	0
May 15, 2019	**26**	**1**	**0**
May 16, 2019	1751	0	0
May 17, 2019	24	0	0
May 20, 2019	137	0	0
May 22, 2019	528	0	0

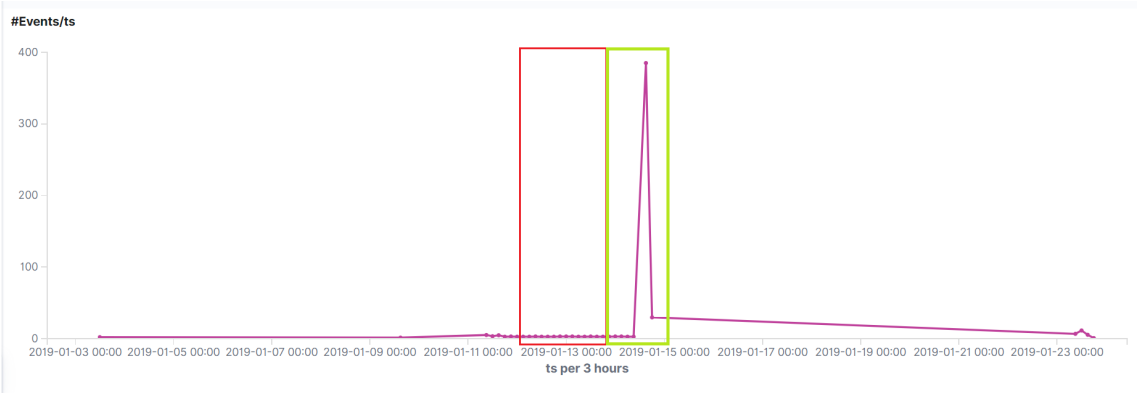
Table 7.11: Example of detected anomalies 4 - Nominal user account with abnormal executable (*Executable_anomaly*) accesses with respect to its past activity (*User_anomaly*).

Date	First Access Time	Last Access Time	#Accesses
Apr 16, 2019	10:05:02	11:52:49	38
Apr 18, 2019	09:19:32	10:32:03	41
Apr 24, 2019	11:07:17	13:27:07	61
Apr 26, 2019	12:18:10	17:38:20	117
Apr 29, 2019	14:28:21	15:34:30	31
May 2, 2019	14:50:26	19:22:50	140
May 3, 2019	15:16:31	18:35:07	75
May 8, 2019	12:07:55	14:17:24	98
May 9, 2019	14:28:58	15:54:02	105
May 10, 2019	15:09:26	16:24:32	57
May 15, 2019	12:49:39	13:52:30	35
May 20, 2019	13:47:30	15:18:05	6
May 23, 2019	**18:13:50**	**19:30:01**	**49**
May 30, 2019	12:24:18	15:21:02	64
Jun 2, 2019	14:55:55	16:19:52	24
Jun 3, 2019	**21:31:21**	**22:34:01**	**3**
Jun 6, 2019	07:36:04	16:52:08	152
Jun 18, 2019	11:49:35	13:16:20	122

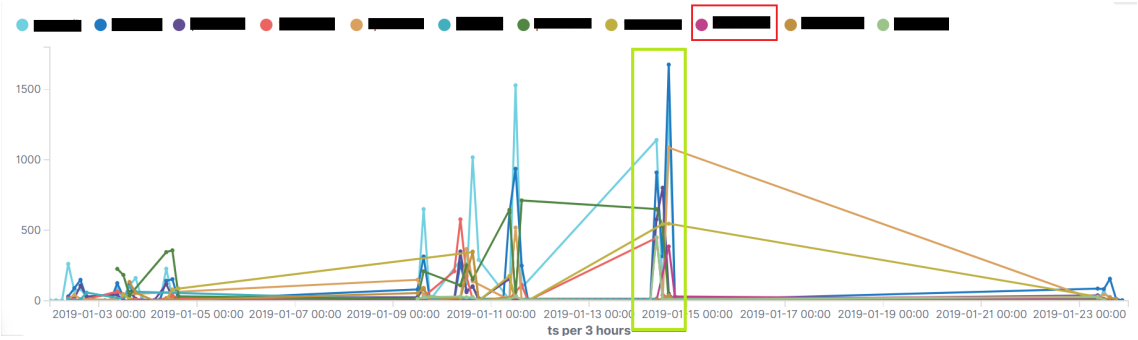
Table 7.12: Example of detected anomalies 5 - Nominal user account with abnormal time accesses (*Hourly_anomaly*) with respect to its past activity (*User_anomaly*).

Another example is shown in Figure 7.17. Here we can see a nominal user account who, similarly to user accounts in the same group (*Group_anomaly*), only accesses on weekdays (*Account_anomaly*). However, it made anomalous accesses in a weekend (*Hourly_anomaly*)

(red). When analyzing this anomaly in 7.15a) it is natural to question 'Why was the following day (green) not detected as an anomaly?'. Examining the behavior of the accounts of the same group on that day, it is easy to see that it is a normal behavior.



(a) User account (two) anomaly days detected (*red*).



(b) Comparison of behavior with user accounts of the same group to justify why the next day peak access volume (*green* in Figure 7.15a) was not detected as an anomaly.

inde	actor	group	date	week_
2695	32783	2	12/01/2019	Sat
2741	32783	2	13/01/2019	Sun

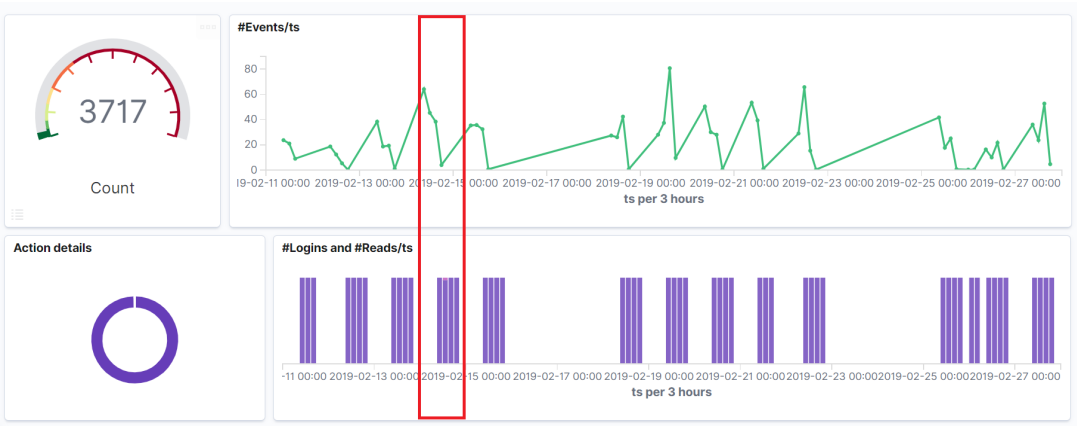
(c) User account accesses on weekend days.

index	actor	group	date	week_
2695	32783	2	12/01/2019	Sat
2741	32783	2	13/01/2019	Sun
3479	1463	2	03/03/2019	Sun
4049	43969	2	02/06/2019	Sun
9117	32359	2	02/02/2019	Sat
9129	32359	2	02/03/2019	Sat
9178	32359	2	09/02/2019	Sat

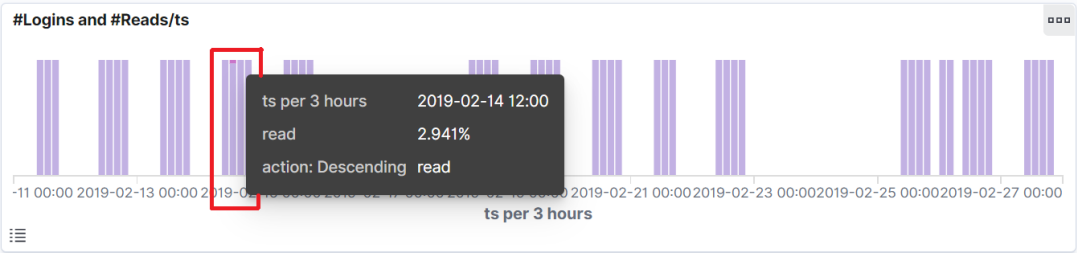
(d) Accesses on weekend days of user accounts belonging to user account group *2*.

Figure 7.15: Example of detected anomalies 6 - Nominal user account with abnormal time (*Hourly_anomaly*) accesses with respect to its past activity (*Account_anomaly*) and its account group activity (*Group_anomaly*).

The following anomalous scenario concerns with the actions performed by a user account, for instance whether it is an abnormal action or not, that is identified by anomaly metric (*Action_anomaly*). Figure 7.16 presents an example of a nominal user account that, similarly to user accounts in the same group (*Group_anomaly*), only performs *login* actions (*Account_anomaly*), but that performs an anomalous *read* action on a day (red).



(a) Screenshot of the temporal dashboard that allows identifying abnormal changes in the type of action taken by the user account on that day (*red*) with respect to its past activity.



(b) Screenshot of the temporal dashboard with zoom on the anomalous day allowing the inspection of abnormal changes in both in the type of action performed and hour of the day (*red*).

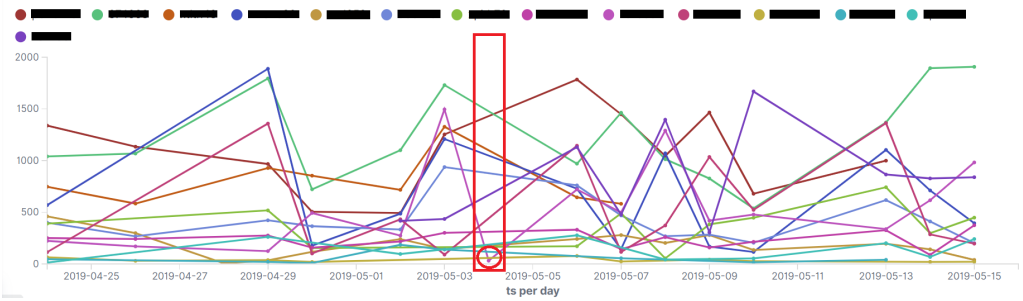
(c) Comparison of behavior with user accounts of the same group validating that *read* actions are abnormal with respect with the user account and its group activity history.

Figure 7.16: Example of detected anomalies 7 - Nominal user account that performed abnormal actions (*Action_anomaly*) with respect to its past activity (*Account_anomaly*) and its account group activity (*Group_anomaly*).

Another type of anomalous scenario can be given by combinations of anomaly metrics such as a user account with abnormal time (*Hourly_anomaly*), location (*Location_anomaly*) and range type (*Range_type_anomaly*) accesses with respect to its past activity (*Account_anomaly*) and/or its account group activity (*Group_anomaly*). A first example of this type

is shown in Figure 7.17 where we can see a nominal user account with:

- Accesses on an anomalous day (*Hourly_anomaly*) with respect to its account group (*Group_anomaly*) in 7.17a, since none of them access on that day (red); and
- Accesses from an anomalous location (*Location_anomaly*) and range type (*Range_type_anomaly*) (red) compared to its past activity (*Account_anomaly*) in 7.17b.



(a) Daily access volume of user accounts belong to the same group allowing to confirm that the user account was the only one of the group with accesses on that day.

(b) Screenshot of the temporal dashboard that allows identifying abnormal changes in user account activity on that day (*red*) with respect to its past activity.

Figure 7.17: Example of detected anomalies 8 - Nominal user account with abnormal time (*Hourly_anomaly*), location (*Location_anomaly*) and range type (*Range_type_anomaly*) accesses with respect to its past activity (*Account_anomaly*) and its account group activity (*Group_anomaly*).

Another example is shown in Figure 7.18. Although the application user account accesses on weekends, as we can see in 7.18a, the anomaly detected refers to an anomalous access made from an unknown location (*Location_anomaly*) and range type (*Range_type_anomaly*) and at a late hour (*Hourly_anomaly*) (red) with respect to the user account's activity history (*Account_anomaly*) at weekends (green).

inde	actor	group	date	week_d
3	560	0	24/03/2019	Sun
4	560	0	10/03/2019	Sun
13	560	0	26/01/2019	Sat
20	560	0	06/04/2019	Sat
21	560	0	07/04/2019	Sun
23	560	0	13/04/2019	Sat
25	560	0	18/05/2019	Sat
27	560	0	29/06/2019	Sat
28	560	0	03/02/2019	Sun
36	560	0	31/03/2019	Sun
38	560	0	14/04/2019	Sun
39	560	0	20/04/2019	Sat
41	560	0	04/05/2019	Sat
46	560	0	08/06/2019	Sat
56	560	0	15/06/2019	Sat
60	560	0	23/02/2019	Sat
62	560	0	17/02/2019	Sun
63	560	0	30/06/2019	Sun
66	560	0	12/05/2019	Sun
73	560	0	30/03/2019	Sat
75	560	0	02/03/2019	Sat
79	560	0	02/02/2019	Sat

(a) User account accesses on weekend days.

2019-06-08: ts per day

ts per day	ts: Descending	actor_range_type: Descending	actor_location: Descending	Count
2019-06-08	Jun 8, 2019 @ 07:00:02.000	range_type1	location1	1
2019-06-08	Jun 8, 2019 @ 07:01:04.000	range_type1	location1	1

2019-06-15: ts per day

ts per day	ts: Descending	actor_range_type: Descending	actor_location: Descending	Count
2019-06-15	Jun 15, 2019 @ 07:00:01.000	range_type1	location1	1
2019-06-15	Jun 15, 2019 @ 07:01:04.000	range_type1	location1	1

2019-06-22: ts per day

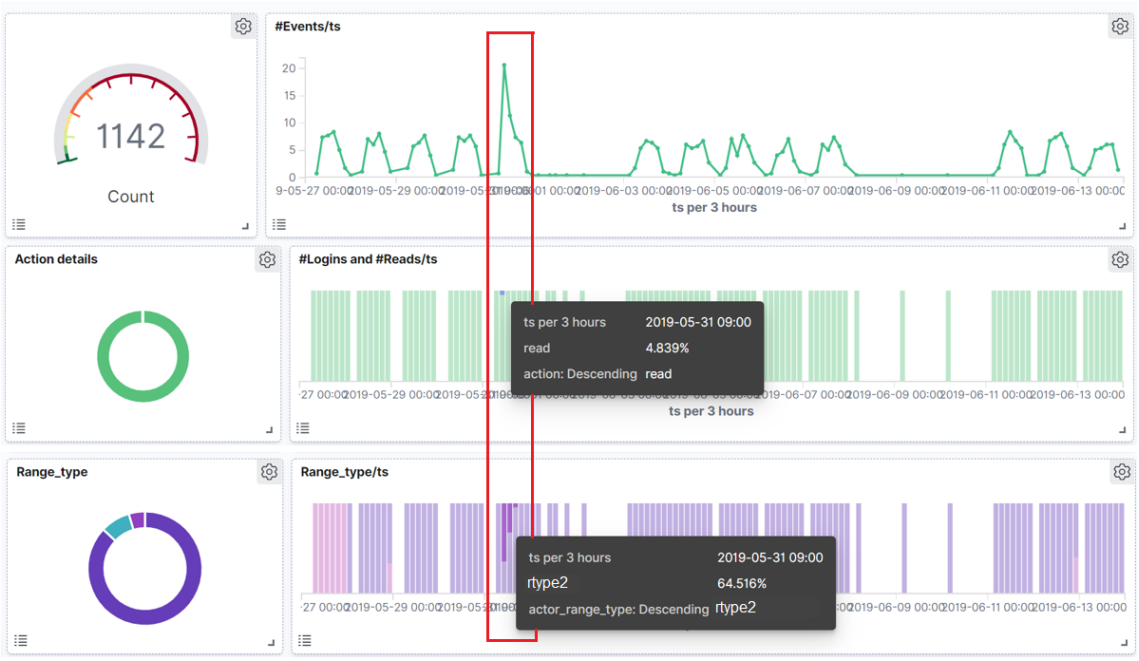
ts per day	ts: Descending	actor_range_type: Descending	actor_location: Descending	Count
2019-06-22	Jun 22, 2019 @ 07:00:01.000	range_type1	location1	1
2019-06-22	Jun 22, 2019 @ 07:01:04.000	range_type1	location1	1
2019-06-22	Jun 22, 2019 @ 23:59:19.000	Missing	Missing	1

(b) User account activity information on some weekend days.

(c) Screenshot of the temporal dashboard that allows identifying abnormal changes in user account activity on that weekend day (*red*) with respect to other weekend days (*green*).

Figure 7.18: Example of detected anomalies 9 - Application user account with abnormal time (*Hourly_anomaly*), location (*Location_anomaly*) and range type (*Range_type_anomaly*) accesses with respect to its activity history (*Account_anomaly*).

The last anomalous scenario concerns with the anomalous actions performed by a user account from an anomalous range type and/or location. Figure 7.19 presents an example of an application user account that, similarly to user accounts in the same group (*Group_anomaly*), only performs *login* actions and has range type *rtype3* (*Account_anomaly*), but that performs an anomalous *read* action with an anomalous range type *rtype2* on a day (red).



(a) Screenshot of the temporal dashboard that allows identifying abnormal changes in both user account range type and type of action performed on that day (*red*) with respect to its past activity.

action: Descending	Count
login	9513
read	3

(b) Frequency table of actions performed by the user account in the six-month period considered.

(c) Comparison of behavior with user accounts of the same group validating that read actions are abnormal with respect with the user account and its group activity history.

Figure 7.19: Example of detected anomalies 10 - Application user account with abnormal action (*Action_anomaly*) and range type (*Range_type_anomaly*) accesses with respect to its past activity (*Account_anomaly*) and its account group activity (*Group_anomaly*) .

7.8 Discussion

Due to the difficulty of the insider threat detection problem, during the development of this project several decisions regarding the best methodology and tools to be adopted in each component had to be made. From the work developed, major challenges faced and strategies created to overcome them are highlighted as follows.

In terms of Data Processing, we had to find an approach to overcome the massive challenge of integrating a dataset with large volume and variety into a standardized dataset in chronological order without exceeding 16GB of available memory. Additionally, we had many multivariate unordered categorical features, some of them with values that were manually created by humans, which leads to some spelling errors and distinct representations for the same value. Besides that, all these issues had to be solved preserving the time series nature of data without loss of information relevant to the problem. The selection of the best features and feature values to consider was carried out with the help of Altice's domain experts. The approach developed divides the time range considered into several time intervals that are iteratively extracted, parsed through a Data Cleaning module, aggregated, and integrated into a structured dataset with user account's daily activity. A good balance between speed, memory and disk usage, and accuracy has been successfully achieved.

Aggregation and feature extraction of the dataset with the purpose of reducing its size and covering all behavioral and contextual information relevant to the problem is one of the biggest challenges of this project. During the development of this project we were faced with the requirement of creating two types of datasets: one for the characterization of the accounts, and consequently detection of anomalous accounts, and other for anomaly detection. In the feature engineering component for creating the first dataset, we were faced with the challenge of representing the daily behavior of each user account in several domains by a single vector. The approach adopted was to aggregate by user account and create eight metrics summarizing each feature's daily behavior. On the other hand, in the feature engineering component for anomaly detection, the challenge was to integrate account's and account group's history information with the account's behavior on that day. To solve this problem, we have developed an extended version of the RobustScaler which centers and scales the dataset taking into account all user accounts' behavioral history, the user account's behavioral history and the behavioral history of all user accounts in the same account group.

Another issue is related to the lack of a clear notion of which are the more suitable detection approaches and algorithms for this purpose. Therefore, multiple machine learning-based Anomaly Detection algorithms were trained and compared.

Another massive challenge was the complex, domain expert dependent and time-consuming task of evaluating the performance of the Anomaly Detection algorithms by virtue of the lack of ground truth and evaluation methods. The results were analyzed and evaluated

against both the injected anomalies and the identified normal instances. Given the volume of the dataset and complexity of the problem, this task requires the involvement of security analysts. The main handicap of this module is that the number of classified instances is not enough to evaluate each algorithm automatically and efficiently. A possible solution to this problem could be using a set of past attacks to evaluate performance.
Although there is no classification and knowledge of all anomalous scenarios and events inserted in the dataset, and consequently the optimal solution cannot be ensured, the overall results achieved show that a large number of anomalies corresponding to distinct types of anomalies have been detected without requiring prior knowledge about them and that its classification leads security analysts to an easy and quick identification of both the anomalous event and its cause.
One of the great advantages of the methodology developed in this project is related to a potential generalizability, as we developed a robust and flexible system capable of adapting to the dynamic nature of accounts and entities activities, regardless the data and application considered.

Chapter 8

Conclusion and Future Work

In this master's project an anomaly detection approach was proposed and implemented to develop a robust automated system for insider threat detection. This project fulfilled its goals of being capable of process and analyze a large collection of activity log data and detect anomalies that can be indicative of insider threat behavior. During the user account behavior modeling, those log records were iteratively cleaned, processed, and integrated into a structured dataset with user account's daily activity. Additionally, by clustering the user account profiles created through multiple aggregation and ratio features, it was possible to build group-based profiles of user accounts with similar behavior. Therefore, from each account daily observation, the system developed a centered and scaled feature set based on all user accounts' behavioral history, on the user account's behavioral history and the behavioral history of all user accounts in the same group. Multiple machine learning-based anomaly detection algorithms were trained on this dataset and evaluated through the injection of different types of scenarios evocative of insider threats.

Experimental results show that the proposed approach encompassing contextual information guarantees robust (high precision) and efficient detection of distinct types of anomalies without requiring prior knowledge about them. The enrichment of the model with anomaly metrics and visualization dashboards also show good results allowing security analysts to identify the anomalous event as well as its cause easily and quickly. While not yet ready to take the approach to production, the results achieved are very promising and encouraging, mainly due to the difficulty of the insider threat detection problem.

As expected, there are some limitations in the work developed in this project, which led us to future research directions. Firstly, since not all anomalies detected will necessarily be an insider threat, the system should be able to distinguish the malicious anomalies from the legitimate ones and associate a risk score/rank to each anomaly according to the threat that it poses. Secondly, with the help of SOC analysts, add a wider range of scenarios for evaluation as well as evaluate some anomalies detected to refines the parameters of the model minimizing the false positive rate. Besides that, improve the prediction performance by considering a combination of these algorithms, more precisely, an unsu-

pervised ensemble-based anomaly-detection approach. Lastly, the model can be enriched with more contextual information about each account, such as the department to which it belongs, the machines it usually accesses, and its risk.
There are already ongoing experiences considering more recent time periods and new application ecosystems whose in-depth study may be also relevant to the business enabling us to further validate our current conclusions and to extend the system in the future directions presented in order to improve the overall performance.

Appendix A

Hyper-parameter Tuning for User Account Characterization

Tables A.1 and A.2 show the parameterization explored for k-Means and Spectral Clustering, respectively.

Table A.1: K-Means Parameter Values.

Parameter	**Values**
n_clusters	range(2,21,1) = {2, 3, 4, 5 ,6 ,..., 18, 19, 20}
init	{'k-Means||', 'k-Means++'}
oversampling_factor	default 2
max_iter	3 000
init_max_iter	200
tol	default 0.0001
algorithm	default 'full'
random_state	0

Table A.2: Spectral Clustering Parameter Values.

Parameter	**Values**
n_clusters	range(2,21,1) = {2, 3, 4, 5, 6, ..., 18, 19, 20}
gamma	{1/(num_features * X.var()), 1/num_features, 0.5, 1}, where num_features = len(X[0])
affinity	default 'rbf'
assign_labels	default 'kmeans'
degree	None
coef0	None
random_state	0
n_jobs	-1

Continued on the next page

Table A.2: Spectral Clustering Parameter Values (Cont.).

Parameter	**Values**
n_components	{num_clusters, 25, 100, num_samples - 1}, where num_samples = len(X)
kmeans_params	default None

The parameterization that reached the most suitable clustering is presented in Table A.3.

Table A.3: Spectral Clustering Parameter Values that achieved better results.

Parameter	**Value**
n_clusters	4
gamma	1/n_features = 1/22
affinity	default 'rbf'
assign_labels	default 'kmeans'
degree	None
coef0	None
random_state	0
n_jobs	-1
n_components	n_clusters = 4
kmeans_params	default None

Appendix B

Hyper-parameter and Weight Assignment of Features Tuning for Anomaly Detection

The range of values explored for each parameter of the compared anomaly detection algorithms are presented in Tables B.1, B.2, B.3, B.4, and B.5. Additionally, Table B.6 shows the range of features weights investigated for each feature group.

Table B.1: k-Nearest Neighbor for Anomaly Detection Parameter Values.

Parameter	**Values**
contamination	{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17}
n_neighbors	{151, 275, 400, 524, 648, 772, 896, 1020, 1144, 1269, 1393, 1517, 1770, 2023, 2276}
method	{'largest', 'mean', 'median'}
radius	default 1.0
algorithm	default 'auto'
leaf_size	{30, 75, 150, 250, 500}
metric	default 'minkowski'
p	{1, 2}
metric_params	default None
n_jobs	-1

Table B.2: One-Class Support Vector Machines Parameter Values.

Parameter	**Values**
kernel	{'rbf', 'poly', 'sigmoid'}
nu	{0.01, 0.11, 0.22, 0.33, 0.44, 0.55, 0.66, 0.77, 0.88, 0.99}
degree	{1, 2, 3, 4, 5}

Continued on the next page

Table B.2: One-Class Support Vector Machines Parameter Values (cont.).

Parameter	**Values**
gamma	{'scale', 'auto', 0.01, 0.25, 0.5, 0.75, 1, 1.75, 2.5}
coef0	{0, 1, 2, 3}
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17}

Table B.3: Local Outlier Factor Parameter Values.

Parameter	**Values**
n_neighbors	{151, 275, 400, 524, 648, 772, 896, 1020, 1144, 1269, 1393, 1517, 1770, 2023, 2276}
algorithm	default 'auto'
leaf_size	{30, 75, 150, 250, 500}
metric	default 'minkowski'
p	{1, 2}
metric_params	default None
contamination	{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17}
n_jobs	-1

Table B.4: Clustering Based Local Outlier Factor Parameter Values.

Parameter	**Values**
n_clusters	list(range(5,100,1)) + list(range(100,201,5)) = {5, 6, 7, ..., 98, 99, 100, 105, 110, ..., 190, 195, 200}
contamination	{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17}
clustering_estimator	default None
alpha	{0.51, 0.56, 0.62, 0.673, 0.727, 0.782, 0.836, 0.89, 0.945, 0.999}
beta	{1, 4, 8, 11, 15, 18, 22, 25, 29, 32, 36, 39, 43, 46, 50}
use_weights	{True, False}
check_estimator	{True, False}
random_state	0
n_jobs	-1

Table B.5: Isolation Forest Parameter Values.

Parameter	**Values**
n_estimators	{250, 300, 350, 400, ..., 900, 950, 1000, 1 250, 1 500, 2 000, 2 500}
max_samples	{'auto', 750, 1 500, 2 000, 2 750, 3 500, 5 000, 7 500, 10 000, 15 000}
contamination	{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17}
max_features	{30, 31, 32, 33, ..., 49, 50, 51}
bootstrap	{True, False}
n_jobs	-1
behaviour	{'old', 'new'}
random_state	0
verbose	default 0

Table B.6: Range of features weights investigated for each feature group.

Parameter	**Values**
w_{group}	{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9}
w_{user}	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
w_{all}	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
Condition	$w_{group} + w_{user} + w_{all} = 1$
w_{action}	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
$w_{location}$	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
w_{range_type}	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
$w_{executable}$	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
w_{number}	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
w_{hour}	{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45}
Condition	$w_{action} + w_{location} + w_{range\_type} + w_{executable} + w_{number} + w_{hour} = 1$

Tables B.7 and B.8 present the combination of feature weights and parameters that achieved the best result, respectively.

Table B.7: Combination of features weights that achieved the best result.

Weight	**Value**
w_{group}	0.50
w_{user}	0.35
w_{all}	0.25
w_{action}	0.15
$w_{location}$	0.20

Continued on the next page

Table B.7: Combination of features weights that achieved the best result (cont.).

Weight	**Value**
w_{range_type}	0.15
$w_{executable}$	0.10
w_{number}	0.20
w_{hour}	0.20

Table B.8: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result.

Parameter	**Value**
kernel	'rbf'
nu	0.11
degree	-
gamma	'auto'
coef0	-
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.15

The range of values explored for each parameter of the compared anomaly detection algorithms follows the one presented above with the exception of the range of parameters considered for max_features parameter in the Isolation Forest algorithm. As such, Table B.9 presents the range of parameters used in max_features for each anomaly metric.

Table B.9: Range of values considered in max_features of the Isolation Forest algorithm for each anomaly metric.

Anomaly Metric	**max_feature Values**
Action_anomaly	{6, 7, 8, 9, 10, 11, 12}
Location_anomaly	{1, 2, 3, 4}
Range_type_anomaly	{13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26}
Executable_anomaly	{7, 8, 9, 10, 11, 12, 13, 14}
Number_anomaly	{1, 2, 3}
Hourly_anomaly	{7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
Account_anomaly	{9, 10, 11, 12, 13, 14, 15, 16}
Group_anomaly	{10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
All_anomaly	{6, 7, 8, 9, 10, 11, 12, 13, 14, 15}

Tables B.10, B.12, B.13, B.14, B.16, B.17, and B.18 present the combination of One-Class Support Vector Machines Algorithm parameters that achieved the best result for Action_anomaly, Range_type_anomaly, Executable_anomaly, Hourly_anomaly, User_anomaly, Group_anomaly, and All_anomaly, respectively. Additionally, Tables B.11 and B.15 show the combination of Isolation Forest Algorithm parameters that achieved the best result for Location_anomaly and Number_anomaly, respectively.

Table B.10: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Action_anomaly.

Parameter	**Value**
kernel	'rbf'
nu	0.88
degree	-
gamma	1.75
coef0	-
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.01

Table B.11: Combination of Isolation Forest algorithm parameters that achieved the best result for Location_anomaly.

Parameter	**Value**
n_estimators	250
max_samples	2 750
contamination	0.08
max_features	3
bootstrap	True
n_jobs	-1
behaviour	'old'
random_state	0
verbose	default 0

Table B.12: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Range_type_anomaly.

Parameter	Value
kernel	'rbf'
nu	0.22
degree	-
gamma	0.5
coef0	-
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.1

Table B.13: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Executable_anomaly.

Parameter	Value
kernel	'rbf'
nu	0.44
degree	-
gamma	'scale'
coef0	-
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.02

Table B.14: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Hourly_anomaly.

Parameter	Value
kernel	'rbf'
nu	0.01
degree	-
gamma	0.01
coef0	-
tol	default 0.001

Continued on the next page

Table B.14: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Hourly_anomaly (cont.).

Parameter	**Value**
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.04

Table B.15: Combination of Isolation Forest algorithm parameters that achieved the best result for Number_anomaly.

Parameter	**Value**
n_estimators	250
max_samples	2 750
contamination	0.05
max_features	2
bootstrap	True
n_jobs	-1
behaviour	'old'
random_state	0
verbose	default 0

Table B.16: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for User_anomaly.

Parameter	**Value**
kernel	'sigmoid'
nu	0.01
degree	-
gamma	1
coef0	1
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.13

Table B.17: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for Group_anomaly.

Parameter	**Value**
kernel	'rbf'
nu	0.44
degree	-
gamma	1
coef0	-
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.15

Table B.18: Combination of One-Class Support Vector Machines algorithm parameters that achieved the best result for All_anomaly.

Parameter	**Value**
kernel	'rbf'
nu	0.11
degree	-
gamma	2.5
coef0	-
tol	default 0.001
shrinking	default True
cache_size	default 200
verbose	default False
max_iter	default -1
contamination	0.07

References

[1] Aymen Abid, Abdennaceur Kachouri, and Adel Mahfoudhi. Outlier detection for wireless sensor networks using density-based clustering approach. *IET Wireless Sensor Systems*, 7(4):83–90, 2017.

[2] Charu C. Aggarwal. Outlier analysis. *Springer*, 2017.

[3] Charu C. Aggarwal and Philip Yu. Outlier detection with uncertain data. *SIAM International Conference on Data Mining*, page 483–493, 2008.

[4] Malik Agyemang, Ken Barker, and Reda Alhajj. A comprehensive survey of numeric and symbolic outlier mining techniques. *Intelligent Data Analysis*, 10(6):521–538, 2006.

[5] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

[6] Mohammad Almseidin, Maen Alzubi, Kovács Szilveszter, and Mouhammd Alkasassbeh. Evaluation of machine learning algorithms for intrusion detection system. 2018.

[7] Khaled Alrawashdeh and Carla Purdy. Toward an online anomaly intrusion detection system based on deep learning. pages 195–200, 2016.

[8] Ahmad Alzghoul and Magnus Lofstrand. Increasing availability of industrial systems through data stream mining. *Computers & Industrial Engineering*, 60(2):195–205, 2011.

[9] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. volume 2431, pages 15–26, 2002.

[10] Aya Ayadi, Oussama Ghorbel, Abdulfattah M. Obeid, and Mohamed Abid. Outlier detection approaches for wireless sensor networks: A survey. *Computer Networks*, 129:319–333, 2017.

[11] Christoph Baur, Benedikt Wiestler, Shadi Albarqouni, and Nassir Navab. Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. *arXiv preprint arXiv:1804.04488*, 2018.

[12] Victor Berger. *Anomaly detection in user behavior of websites using hierarchical temporal memories: Using machine learning to detect unusual behavior from users of a web service to quickly detect possible security hazards*. PhD thesis, KTH Royal Institute of Technology, 2017.

[13] Daniel S. Berman, Anna L. Buczak, Jeffrey S. Chavis, and Cherita L. Corbett. A survey of deep learning methods for cyber security. *Information*, pages 1–35, 2019.

[14] Brock Bose, Bhargav Avasarala, Srikanta Tirthapura, Yung-Yu Chung, and Donald Steiner. Detecting insider threats using radish: A system for real-time anomaly detection in heterogeneous data streams. *IEEE Systems Journal*, pages 1–12, 01 2017.

[15] Markus Breunig, Hans-Peter Kriegel, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 2000.

[16] Dawn Cappelli, Andrew L. Moore, and Randall Trzeciak. The cert guide to insider threats: How to prevent, detect, and respond to information technology crimes (theft, sabotage, fraud). *Addison-Wesley*, 2012.

[17] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407v2*, 2019.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3), 2009.

[19] Jinghui Chen, Saket Sathe, Charu C. Aggarwal, and Deepak Turaga. *Outlier detection with autoencoder ensembles*, pages 90–98. 2017.

[20] Malcolm Corney, George Mohay, and Andrew Clark. Detection of anomalies from user profiles generated from system logs. volume 116, pages 23–32, 2011.

[21] Abhinandan Das, Mayur Datar, Ashutosh Garg, and ShyamSundar Rajaram. Google news personalization: Scalable online collaborative filtering. volume 116, page 271–280, 2007.

[22] Ethan Dereszynski and Thomas G. Dietterich. Spatiotemporal models for anomaly detection in dynamic enviromental monitoring campaigns. *ACM Transactions on Sensor Networks*, 8(1):3:1–3:26, 2011.

[23] Rémi Domingues, Maurizio Filippone, Pietro Michiardi, and Jihane Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Elsevier*, 74, 2018.

[24] Francis Ysidro Edgeworth. On discordant observations. *Philosophical Maganize*, 23(5):364–375, 1887.

[25] Hoda Eldardiry, Sricharan Kumar, Juan Liu, John Hanley, Bob Price, Oliver Brdiczka, and Eugene Bart. Multi-source fusion for anomaly detection using across-domain and across-time peer group consistency checks. *International Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications*, 01 2013.

[26] Andrew Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158*, 2015.

[27] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. In *International Conference on Machine Learning*, pages 255–262, 2000.

[28] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Salvatore Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of Data Mining in Computer Security*, 6, 2002.

[29] Filipe Falcão, Tommaso Zoppi, Caio Barbosa Vieira da Silva, Anderson Santos, Baldoino Fonseca, Andrea Ceccarelli, and Andrea Bondavalli. Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection. *34th ACM/SIGAPP Symposium on Applied Computing*, pages 318–327, 2019.

[30] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, 2009.

[31] Gaurang Gavai, Kumar Sricharan, Dave Gunning, John Hanley, Mudita Singhal, and Robert J. Rolleston. Supervised and unsupervised methods to detect insider threat from enterprise social and online activity data. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 6(4):47–63, 2015.

[32] Gaurang Gavai, Kumar Sricharan, Dave Gunning, Robert Rolleston, John Hanley, and Mudita Singhal. Detecting insider threat from enterprise social and online activity data. In *7th ACM CCS International Workshop*, pages 13–20, 10 2015.

[33] Gebeyehu Belay Gebremeskel, Chai Yi, Zhongshi He, and Dawit Haile. Combined data mining techniques based patient data outlier detection for health safety. *International Journal of Intelligent Computing and Cybernetics*, 9(1):42–68, 2016.

[34] Prasanta Gogoi, Dhruba K. Bhattacharyya, Bhogeswar Borah, and Jugal Kalita. A survey of outlier detection methods in network anomaly identification. *The Computer Journal*, 54(4):570–588, 2011.

[35] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS ONE*, 11(4), 2016.

[36] Julie Greensmith, Jamie Twycross, and Uwe Aickelin. Dendritic cells for anomaly detection. *IEEE Congress on Evolutionary Computation*, pages 664–671, 2006.

[37] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[38] Miha Grčar, Dunja Mladenić, and Marko Grobelnik. User profiling for interest-focused browsing history. *CEUR Workshop Proceedings*, 137:99–109, 10 2005.

[39] Manish Gupta, Jing Gao, Charu Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2014.

[40] Adam Hall, Nikolaos Pitropakis, William Buchanan, and Naghmeh Moradpoor. Predicting malicious insider threat scenarios using organizational data and a heterogeneous stack-classifier. In *2018 IEEE International Conference on Big Data*, pages 5034–5039, 12 2018.

[41] Shilin He, Jieming Zhu, Pinjia He, and Michael Lyu. Experience report: System log analysis for anomaly detection. pages 207–218, 2016.

[42] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster based local outliers. *Pattern Recognition Letters*, 24:1641–1650, 2003.

[43] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2), 2004.

[44] Joanna Hu, Baoming Tang, and Derek Lin. Anomalous user activity detection in enterprise multi-source logs. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 797–803, 11 2017.

[45] Sans Institute. Monitoring privileged user actions, 2010.

[46] Wen Jin, Anthony Tung, Jiawei Han, and Wei Wang. Ranking outliers using symmetric neighborhood relationship. pages 577–593, 2006.

[47] Ian Jolliffe. Principal component analysis. 2nd ed. *Springer*, 98, 2005.

[48] Irina Kakanakova and Stefan Stoyanov. Outlier detection via deep learning architecture. pages 73–79, 2017.

[49] Firuz Kamalov and Ho Hon Leung. Outlier detection in high dimensional data. *arXiv preprint arXiv:1909:03681*, 2019.

[50] Heung-Nam Kim, Inay Ha, Kee-Sung Lee, Geun-Sik Jo, and Abdulmotaleb El-Saddik. Collaborative user modeling for enhanced content filtering in recommender systems. *Decision Support Systems*, 51(4):772–781, 2011.

[51] Taewoo Kim, Nam Kyu Park, Honghyun Cho, and Kang. Insider threat detection based on user behavior modeling and anomaly detection algorithms. *Applied Sciences*, 9:4018, 09 2019.

[52] Rash Lashkari, Min Chen, and Ali Ghorbani. A survey on user profiling model for anomaly detection in cyberspace. *Journal of Cyber Security and Mobility*, 8:75–112, 2018.

[53] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *SIAM International Conference on Data Mining*, page 25–36, 2003.

[54] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. page 157–166, 2005.

[55] Duc C. Le and Nur Zincir-Heywood. Machine learning based insider threat modelling and detection. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1–6, 2019.

[56] Phil Legg, Oliver Buckley, Michael Goldsmith, and Sadie Creese. Automated insider threat detection system using user and role-based profile assessment. *IEEE Systems Journal*, 99:1–10, 06 2015.

[57] Yong Li, Tao Zhang, Yuan Yuan Ma, and Cheng Zhou. Anomaly detection of user behavior for database security audit based on ocsvm. *Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on. IEEE*, pages 214–219, 2016.

[58] Alexander Liu, Cheryl Martin, Tom Hetherington, and Sara Matzner. A comparison of system call feature representations for insider threat detection. pages 340 – 347, 2005.

[59] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. pages 413 – 422, 2009.

[60] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1249–1268, 2016.

[61] Lorenzo Fernández Maimó, Ángel Luis Perales Gómez, Félix J. García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. Self-adaptive deep learning-based system for anomaly detection in 5g networks. *IEEE Access*, 6:7700–7712, 2018.

[62] Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. Mining outliers with ensemble of heterogeneous detectors on random subspaces. *Database Systems for Advanced Applications, Berlin, Germany: Springer*, page 368–383, 2010.

[63] Grant Pannell and Helen Ashman. Anomaly detection over user profiles for intrusion detection. *Australian Information Security Management Conference*, 2010.

[64] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

[65] Ebberth Paula, Marcelo Ladeira, Rommel Carvalho, and Thiago Marzagão. Deep learning anomaly detection as support fraud investigation in brazilian exports and anti-money laundering. pages 954–960, 2016.

[66] Marco A. F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[67] Mariana Pina. Automatic detection of anomalous user access patterns to sensitive data, 2019.

[68] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. *Medical Image Analysis*, 8(3):275–283, 2001.

[69] Utkarsh Porwal and Smruthi Mukund. Credit card fraud detection in e-commerce: An outlier detection approach. *arXiv preprint arXiv:1811.02196*, 2018.

[70] T. Prarthana and Nandyala Gangadhar. User behaviour anomaly detection in multi-dimensional data. 2017.

[71] Marcel Prastawa, Elizabeth Bullitt, Sean Ho, and Guido Gerig. A brain tumor segmentation framework based on outlier detection. *Medical Image Analysis*, 8(3):275–283, 2004.

[72] Sridhar Ramaswamy, Rajeev Rastogi, Kyuseok Shim, and Taejon Korea. Efficient algorithms for mining outliers from large data sets. 2000.

[73] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza Samatova. Anomaly detection in dynamic networks: A survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015.

[74] Farzad Sabahi and Ali Movaghar. Intrusion detection: A survey. *3rd International Conference on Systems and Networks Communications - ICSNC'08. IEEE*, pages 23–26, 2008.

[75] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alexander J. Smola, and Robert C. Williamson. Estimating support of a high-dimensional distribution. 1999.

[76] Nauman Shahid, Ijaz Haider Naqvi, and Saad Bin Qaisar. Characteristics and classification of outlier detection techniques for wireless sensor networks in harsh environments: A survey. *Artificial Intelligence Review*, 43(2):193–228, 2015.

[77] Madhu Shashanka, Min-Yi Shen, and Jisheng Wang. User and entity behavior analytics for enterprise security. *2016 IEEE International Conference on. IEEE*, page 1867–187, 2016.

[78] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):631–645, 2007.

[79] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. pages 675–684, 01 2004.

[80] Li Sun, Steven Versteeg, Serdar Bozta¸s, and Asha Rao. Detecting anomalous user behavior using an extended isolation forest algorithm: An enterprise case study. *arXiv preprint arXiv:1609.06676v1*, 2016.

[81] Priyanga Dilini Talagala, Rob J. Hyndman, and Kate Smith-Miles. Anomaly detection in high dimensional data. *arXiv preprint arXiv:1908:04000*, 2019.

[82] Jinita Tamboli and Madhu Shukla. A survey of outlier detection algorithms for data streams. *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 3535–3540, 2016.

[83] Jian Tang, Zhixiang Chen, Ada Fu, and David Cheung. Enhancing effectiveness of outlier detections for low density patterns. pages 535–548, 2002.

[84] Lionel Tarassenko, Paul Hayton, Nicholas Cerneaz, and Michael Brady. Novelty detection for the identification of masses in mammograms. *4th International Conference on Artificial Neural Networks*, pages 442–447, 1995.

[85] Haystax Technology. New haystax technology survey shows most organizations ill-prepared for insider threats, 2017. Last accessed 22 November 2019.

[86] Senator Ted, Henry Goldberg, Alex Memory, William Young, Bradley Rees, Robert Pierce, Daniel Huang, Matthew Reardon, David Bader, and Edmond Chow. Detecting insider threats in a real corporate database of computer usage activity. pages 1393–1401, 01 2013.

[87] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. *The AAAI-17 Workshop on Artificial Intelligence for Cyber Security*, pages 224–231, 2017.

[88] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. volume 9, pages 2579–2605, 2008.

[89] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. Progress in outlier detection techniques: A survey. *IEEE Access*, 2019.

[90] Xi Xiangyu, Tong Zhang, Dongdong Du, Guoliang Zhao, Qing Gao, Wen Zhao, and Shikun Zhang. Method and system for detecting anomalous user behaviors: An ensemble approach. pages 263–307, 2018.

[91] Feng Xue, Weizhong Yan, Nicholas Roddy, and Anil Varma. Operational data based anomaly detection for locomotive diagnostics. *International Conference on Machine Learning*, pages 236–241, 2006.

[92] William Young, Alex Memory, Henry Goldberg, and Ted Senator. Detecting unknown insider threat scenarios. volume 2014, pages 277–288, 05 2014.

[93] Bin Zhang, Chris Sconyers, Carl Byington, Romano Patrick, Marcos Orchard, and George Vachtsevanos. Anomaly detection: A robust approach to detection of unanticipated faults. pages 1 – 8, 11 2008.

[94] Haibin Zhang, Jiajia Liu, and Cheng Zhao. Distance based method for outlier detection of body sensor networks. *EAI Endorsed Transactions on Wireless Spectrum*, 2:e4, 01 2016.

[95] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: a python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20:1–7, 2019.

[96] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5):363–387, 2012.